APPLICATIONS OF FUZZY LOGIC TO SOFTWARE METRICS MODEL

YUHANIM HANI BINTI YAHAYA

A thesis submitted to the
Faculty of Computer Science and Information Technology, University Malaya
in partial fulfillment of the requirements
for the degree of Master of Computer Science

FACULTY OF
COMPUTER SCEINCE AND INFORMATION TECHNOLOGY

UNIVERSITI MALAYA

FEBRUARY 2000

DECLARATION

I certify that this thesis submitted for the degree of Masters is the result of my own research, except where otherwise acknowledged, and that this thesis (or any part of the same) has not been submitted for higher degree to any other university of institution.

Signed:.....

Yuhanim Hani Yahaya

Date: 4 FEBRUARY 2000

ABSTRACT

This thesis investigates the use of fuzzy logic approach in software metrics application. Software metrics defines a standard way of measuring the properties of software products, development processes and resources. The most common application of software metrics is the software cost estimation where it predicts the effort required for completing certain stages of a software development life cycle. The ability to obtain an accurate effort prediction is essential for the cost estimation process, as it helps a project manager to specify the efforts needed for project development. In relation to this, cost estimation models such as COCOMO (Constructive Cost Model) , Function Points and SLIM (Software Life-Cycle In Management) which used specified equations for estimating development effort have been proposed. However, these models suffer several limitations in terms of the inputs. Inputs to these models may include experience of the programmer, the required reliability of the software, complexity of the project and an estimate of the project size. These inputs are subjective (non-numerical) and thus requiring expert knowledge. Some of these inputs are confusing and are not known with reasonable degree of certainty until the project is completed. In an attempt to overcome this problem and to fulfil the needs, fuzzy logic has been studied for effort prediction. Fuzzy logic is a form of logic that deals with subjective and uncertainty values. It allows expert knowledge to determine the input values by using the linguistic terms instead of mathematical equations. Project managers are in fact able to classify the inputs by using linguistic terms such as "High level of project complexity" and "Size of the project is medium". The **F**uzzy **L**ogic **E**ffort **P**rediction (**FLEP**) program is produced in an attempt to model the fuzzy logic approach in estimating effort development. Using Mean Magnitude of Relative Error (MMRE) measurement, a comparison between fuzzy logic and other techniques have been made. Four cost estimation techniques are compared, COCOMO, Function Points, SLIM and Fuzzy Logic. The results showed that fuzzy logic technique provides better estimation for development effort in the early stage of development life cycle.

ACKNOWLEDGMENT

It is my pleasure to acknowledge many people who contributed to the preparation of this thesis. In particular, I am most indebted to my very helpful and resourceful supervisor Mr. Phang Keat Keong (University of Malaya) who played the most important role in his time sacrificed to discuss, comment, give suggestions and constructive criticisms. The quality of this thesis was improved and organized by his thoughtful responses.

Further, special thanks is also dedicated to the encouraging and perceptive dean of the IT Faculty University Tun Abdul Razak (UNITAR) Prof. Dr. Khairuddin for his genuine support and encouragement. I would also like to thank UNITAR for providing a scholarship, which has made this work possible.

I extend my thanks to Mr. Andrew Gray from Software Metrics Research Laboratory (University of Otago) for providing constant source of knowledge and advice during the course of my thesis.

To all my adorable friends and colleagues for accommodating my time spent researching and writing. I appreciate the moral support and editorial help.

I would also like to thank mum and dad and all my family members for their incorruptible support, patience, love and dedication in assisting me to produce a high quality tesis. I realized that their encouragement and wise advice (wisdom) have taught me that it is not impossible to transform dream into reality with strength and courage. Thank you very much.

Finally, a very special thanks to beloved Nin, who carried more than his responsibility whilst maintaining his love, encouragement, support and understanding.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

COCOMO	Constructive Cost Model
DBMS	Database Management System
EAF	Effort Adjustment Factor
FLEP	Fuzzy Logic Effort Prediction
IDE	Integrated Development Environment
KDSI	Thousands Of Delivered Source Of Instructions
KLOC	Thousands Of Lines Of Source Code
LOC	Lines Of Code
MM	Man Month
MRE	Magnitude Relative Error
PROLOG	Programming In Logic
RE	Relative Error
SLIM	Software Life Cycle Management
SLOC	Source Lines Of Code
TCF	Technical Complexity Factor
UC	Unadjusted Count

CHAPTER 1

INTRODUCTION

Recently, the field of software metrics has gained much popularity in software development. Software metrics deals with the measurement of the software product, process by which it is developed and resources used to develop it. It can effectively measure the complexity of the project, the cost and effort involved in developing the software product. Good metrics should facilitate the capability of predicting process, product or resources, which reflects the quality process to develop the software product within minimal resources. This chapter provides an overview of the project. It also discusses the motivation of the current work, the objective and the scope of the project. An outline of this thesis is given in the end of this chapter.

1.1 Definition Of The Project

Software metrics is measurements of the software development process and product that can be used as independent and dependent variables in models for project management [1]. There are many software metrics applications such as software cost estimation models, productivity models, quality control models, data collection, reliability models, structural and complexity metrics and other metrics [2,3]. Here software cost estimation model can be considered as an important application of the software metrics. It is useful in predicting the effort required for software development. The effort is usually measured in person-months or person-years.

It is important for a project manager to estimate the effort required for the project in the early development life cycle. This will help the project manager to know how many developers are required since it affects resource allocation and project feasibility. Consider a typical system development project faced with targets for both the software to be delivered and the time in

which to deliver it. Those responsible for planning the project require estimates of the effort needed to deliver the software within the targeted time. A critical problem faced by all software project managers is the accuracy of effort estimation. Accurate estimation of software development effort has major implications for the management of software development [2]. A low estimate implies that the software development team will be under pressure to complete the project early and hence, the resulting software may not be fully tested or contains all the required functionality [2]. On the other hand, a high estimate implies that too many resources will be committed to the project. In order to overcome this problem, several cost estimation models such as COCOMO, Function Points and SLIM were developed.

The COCOMO was developed by Boehm [2,3,4,5,8] based on the analysis of 63 software projects. There are three COCOMO models; the Basic model that can be applied when little about the project is known, the Intermediate model that is applied after requirements are specified and the Advanced model that is applied when design is complete. All the three models relate the effort required for software development and source lines of code (SLOC) as a major input as follows:

$$E = a \times (S)^b F$$

where

E is effort in person-months,

S is the size measures in thousands of source lines of code

F is an adjustment factor.

The values of a and b depend on the development mode. There are three modes of development; an organic mode, an embedded mode and semi-detached mode. The organic mode that involves data processing tends to use databases and focus on transactions and data retrieval. The embedded mode contains hardware-based system and real-time software system while the semi-detached mode is somewhere between organic and embedded mode. By selecting an appropriate development mode and applying to the effort equation, COCOMO yields effort estimation.

The SLIM model is based on the Putnam's own analysis of the software life cycle in terms of the Rayleigh distribution of project personnel level versus time. The equation is in the form as follows:

$$\text{Size} = C_k K^{1/3} t^{4/3}_d$$

This equation relates size (measured as lines of code) to several variables: a technology factor, C, total project effort measured in person-years, K (which includes maintenance effort) and elapsed time to delivery, t_d measured in years [4,5,8].

Function Points was developed by Albrecht [3,4,5,7,8,9] in 1979. It is a method of estimating effort by measuring the functionality of a system based on the characteristics of user functionality as follows:

- External inputs (e.g. file name)
- External outputs (e.g. reports, messages)
- Inquiries (interactive inputs needing a response)
- External files or interfaces (file shared with other software systems)
- Internal files (invisible outside the system)

In addition, there are 14 technical complexity factor's characteristics such as data communications, performance, transaction rate, online data entry and others. Each project technical complexity factor is rated on the basis of its degree of influence, which varies from 1 to 5.

Function Points are counted based on the weighting user functionality as follows:

$$FP = UC * (0.65 + 0.01 * TCF)$$

where

UC is the Unadjusted Count

TCF is the Technical Complexity Factor.

The unadjusted count is defined as

$$UC = aI + bO + cE + dL + eF$$

where

I is the number of input types

O is the number of output types

E is the number of inquiry types

L is the number of logical internal files

F is the number of interfaces

a, b, c, d, e, > 0 are the weighting factors.

Inputs to the models discussed so far may include a variety of factors such as the size of the project, the required reliability of the software, use of software tools, data processing and the experience of the programmers. However, these values are vague, imprecise and difficult to define in the early process of the software development life cycle. Input values are uncertain when we are unable to measure their values directly. For example, when using a model such as the Intermediate COCOMO model to estimate effort at the start of a project, the number of lines of code must be estimated and judgement must be used to decide the values for the cost drivers [6,7]. Fifteen cost drivers are used in the Intermediate COCOMO model [6,7] such as software reliability, experience of the programmer and others. The influence of a cost driver on effort is

modeled by multipliers, which either increase or decrease the nominally expected effort. For example, a more experience programmer is expected to reduce effort by 9% [6,7]. Function Points measure also incorporates the influence of the cost drivers. The unadjusted Function Points are multiplied by the degree of influence of the 14 technical complexity factors. These cost drivers value does not offer any clues about how certain or uncertain an estimate is.

Due to the fact that the inputs to the cost estimation model are vague and uncertain, it is proposed that the input to the cost estimation model should be supplemented with fuzzy logic approach. Investigation of fuzzy logic is the core study of this thesis. It is suffice to indicate that fuzzy linguistic variable can capture the subjective, vague and imprecise inputs. A linguistic variable is a variable whose value can be described qualitatively using an expression involving linguistic terms and quantitatively using a corresponding membership function [10]. The linguistic term is useful for communicating knowledge with human beings; whereas membership function is useful for processing numeric input data [10]. In general, the value of a linguistic variable can be a linguistic expression involving a set of linguistic term's modifiers such as "very" or "more or less". For example, a project manager may specify the size of a project by saying that the size of a project is "very small", the size of the project is "more or less small" or the size of the project is "medium". The meaning of such expression is governed by a series of fuzzy if-then rules, which can be used to derive prediction for the effort. The if-then rules will be discussed in the next section.

1.2 Motivation Of Current Work

In spite of much research in developing models for software cost estimation, it appears that the estimates are not associated with significantly greater accuracy [3]. The models that have large number of input parameters are difficult to calibrate, where these input parameters might not be independent. Kitchenham [3] demonstrates that there is a relationship between the two of the COCOMO cost drivers (experience of a programmer and software reliability) based on projects

in the COCOMO data set and points out that the relationships between input parameters will make a model unstable. The cost driver values are usually based on subjective assessment.

Secondly, it is difficult for estimator to make accurate effort estimation. With the existing software cost estimation models, the estimation in the early development life cycle are based on uncertain input values. It is difficult to have exact values when we are unable to measure their values directly. In addition, estimators may not be able to develop a model based on direct measures because suitable historical data is not available [6]. The standard deviation, is a measure of the variation within the historical data set [6]. Unfortunately, this is not a true measure of the uncertainty because the model on which the estimate is based on is incomplete. Consider making an estimate for a project affected by a factor, which is constant for the historical projects on which the model is based. The model will not be able to predict the influence of this factor and there is no guarantee that the actual value for the new project will fall within the range of values predicted. For example, an effort estimate for a new project that has a client-server environment may be based on effort to develop similar systems, which have a host-based environment. If the change in system environment is not identified as a cost driver, its influence will not be taken into account by the estimate. If the change has a significant affect on productivity, the estimate may be quite unreliable [6]. Therefore, the uncertainty of the estimate still depends on both the uncertainty of the model on which it is based and the uncertainty in the model's input.

These problems motivate continuing research into other techniques for software cost estimation. The work presented in this project not only reviews the existing software cost estimation models but also investigates the use of fuzzy logic as an alternative to cost estimation. The use of fuzzy logic seems to be suitable for using the expert judgment from the project managers. A project manager may specify that a project will have a large number of entities and a small number of files [1]. These can be represented as fuzzy variables and fuzzy rules in order to derive the prediction for the output, such as the project effort. The use of fuzzy variables allows us to represent the input values in terms of linguistic value; small, medium or large rather than numerical values; 1 or 0. The linguistic values are vital to model the uncertainty associated with the influence of input values on effort. Consider the size and the duration of the project as the

input to the effort estimation process. These inputs are the linguistic variables, which may have the linguistic values; "large", "medium", "small", "long" and "short". Typically, these values are represented by a range of values in the fuzzy sets known as membership functions as depicted in figure 1.1.

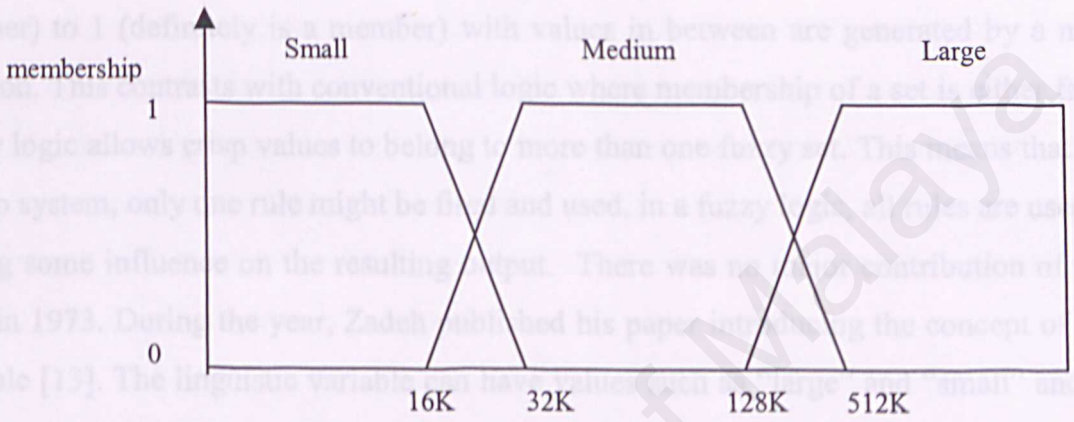

Figure 1.1 Fuzzy Sets Representing Project Size

Inference from a set of fuzzy rules involves fuzzification of the conditions of the rules, then propagating the confidence factors (membership values) of the conditions to the conclusions of the rules. Consider the following if-then rule:

IF project size is large AND project duration is short, THEN effort is high

Inference from the above rule involves looking up the membership value of the condition 'project size is large'- given the source lines of code and the membership value of 'project duration'- given the required time to complete. In order to assign the outcome 'effort is high', minimum membership value of all the conditions must be considered.

Fuzzy logic approach as an alternative cost estimation model, will improve several limitations. The project manager can classify the vague input values into fuzzy variables by using their expert judgement. Another advantage of using the fuzzy logic approach is that it reduces the number of

parameters in the estimation process and uses only appropriate data required for measurement. Certain inputs such as data transaction, data communication and turnaround time are not required in the earlier estimation.

Fuzzy logic [10,11,12,13] was incepted by Lofti Zadeh at the University Of California, Berkeley in 1965. Fuzzy logic is based on the principle that every crisp value belongs to all relevant fuzzy sets to various extents, called the degrees of membership. These range from 0 (definitely not a member) to 1 (definitely is a member) with values in between are generated by a membership function. This contrasts with conventional logic where membership of a set is either false or true. Fuzzy logic allows crisp values to belong to more than one fuzzy set. This means that whereas in a crisp system, only one rule might be fired and used, in a fuzzy logic, all rules are used with each having some influence on the resulting output. There was no major contribution of fuzzy logic until in 1973. During the year, Zadeh published his paper introducing the concept of a linguistic variable [13]. The linguistic variable can have values such as "large" and "small" and the values assigned to each linguistic variable are fuzzy sets.

It is important to note that there are few numbers of ongoing fuzzy logic research projects in software metrics [1,14,15,32]. Much of the fuzzy logic work to date has been concerned with the development effort estimation [1]. Accuracy and consistencies in the prediction of software development effort early in the software process has long been a goal of those involved in software metrics research. Effort estimation using fuzzy logic model allowing less precise estimation from project managers when they provide early estimates of project size and complexity. Instead of estimating that a project will contain 5000 function points, the managers can describe the system as *medium-large*. It is stressed here that such labels need to be calibrated to the organisation itself, this project may be *very large* to another.

Another application of fuzzy logic was modeling the software development process by using an expert simulation system [14]. The software development process is characterized by certain types of factors that are imprecise and vague such as product factors (size and complexity), situational factors (stability of project requirements and budgetary) and environmental factors (availability of software tools). In an attempt to more effectively model the software development

process, these factors are treated as fuzzy variables. The fuzzy variables in the expert simulation models are handled by an input expert system, which uses fuzzy logic. This expert system is designed to check the consistency of input variables and to help managers of software projects to plan and manage the life cycle of a development process effectively.

Applications of fuzzy logic to software metrics also include the computer software source code authorship analysis [15]. There are three main areas in authorship analysis, namely, author discrimination, author characterization and similarity detection. While a large number of metrics has been proposed for this task, there is a difficulty with capturing certain types of information in terms of quantitative measurement [15]. Fuzzy logic linguistic variables was proposed in order to capture more subjective elements of authorship such as the degree to which comments match the actual source code's behavior. These variables avoid the need for complex and subjective rules, replacing these with an expert's judgement.

A significant proportion of research on developing predictive software metrics has focused on using non-traditional methods, such as neural networks and fuzzy logic [32]. Three broad areas of concern that can be cited with regard to software development predictive models are the difficulties of data sets, acceptability and validation of models and generalisability. The most obvious strength of using the fuzzy logic methods is its fuzzy linguistic mappings. A fuzzy linguistic mapping is a highly intuitive model that can be created by anyone, without training. It is a mapping between linguistics terms, such as "very small", attached to variables.

1.3. Scope of the Work

The scope of this work is mainly to develop a fuzzy logic program that will determine the effort required for a particular project development. In this dissertation, we particularly focus on how a fuzzy logic model could be used to predict the effort required. Existing models such as COCOMO, Function Points and SLIM are extensively explored and a comparison of these

existing models and the fuzzy logic model is conducted to determine the modeling power of these models.

The literature on software cost estimation model is quite extensive. In the existing models the underlying problem is on the fact that these models possess uncertain input values and some of the input are not required in the early estimation, for example turnaround time, software reliability and so on. In contrast, fuzzy logic model promises a more reliable measure in the effort estimation. Linguistic approach used by the fuzzy logic model can successfully overcome the uncertainty of input value. Generally, the size and complexity of project form the linguistic variables and the linguistic values of these linguistic variables can either be small, medium or large (size) and low, medium and high (complexity). Consequently, the vague and uncertain values of input can be better predicted.

In particular, we develop a program called **F**uzzy **L**ogic **E**ffort **P**rediction (FLEP) program using Amzi! Prolog [40]. The FLEP uses three inputs: size, complexity and the duration of the project. The values of these inputs are then fuzzified into fuzzy sets to capture the uncertain values and to form a fuzzy rule. Inference of a set of fuzzy rules, which involve fuzzification, would determine the effort values. Finally, the effort values will again be defuzzified to get a single crisp value.

In addition, we determine and discuss the estimation accuracy of the fuzzy and non-fuzzy effort prediction techniques in terms of Mean Magnitude Relative Error ($\overline{\text{MRE}}$). Magnitude Relative Error (MRE) [1,7,8] is a measure of the difference between the actual values of the input values (V_A) and the predictions of the model (V_F):

$$\mathbf{MRE} = \frac{|V_A - V_F|}{V_A}$$

The $\overline{\text{MRE}}$ is therefore the mean value for this indicator over all observations in the sample. For software development, relative errors are often a better judgement of the model's accuracy.

1.4 Overview of the Dissertation

The literature review on software metrics and software cost estimation are presented subsequently in Chapter 2 and Chapter 3. These are the basis for analyzing the cost estimation research. Chapter 4 will review the concept of fuzzy logic concept. More generally, it will explain the notion of fuzzy set, linguistic variables and fuzzy rules. The reviews on the use of fuzzy logic as an alternative in cost estimation are presented in Chapter 5. Chapter 6 will describe the performance of the fuzzy logic program in predicting effort. The result of the analysis of the comparison between fuzzy logic and other cost estimation models is presented in Chapter 7. Finally, the conclusion summarizes the work presented in this project, comments on the areas that this project has not addressed and issues relating to the use of fuzzy logic in cost estimation.

CHAPTER 2

SOFTWARE METRICS

There has been much research in recent years about the role of software metrics in development process. Research indicated that software metrics can help achieve better management process. For example, the predictions of completion date made early in the project allowed the project management to predict and avoid possible serious schedule problem. This chapter introduces the notion of software metrics and explains its relevance to software development process including the fundamentals of measurement. The definitions of software metrics and the properties of good metrics together with the classifications, the scope and the usage of software metrics form the other part of the major discussion in this chapter. We intend to describe the use of software metrics as control metrics and predictor metrics. Finally, a proposed framework for prediction, which can be used in software metrics, is explored.

2.1 Software Measurement

We use measurement in everyday life. In economic systems, measurement is used to determine the price and pay increment. Measurement in medical systems, on the other hand, enables us to diagnose specific illness [3]. According to Fenton [3]

> *Measurement* is the process by which numbers or symbols are assigned to attributes of entities in the real world in such a way as to describe them according to clearly defined rules.

Therefore, measurement is concerned with the capturing of information about attributes of entities. An entity may be an object such as a book, a chair or an event of a software project. The attribute in the other hand, is the feature or property of the entity, which we are interested in. By referring to the viewpoint of managers and engineers we will be able to derive the objectives of the measurement. These views are shown in following:

Managers:

1. Need to measure the cost of various processes within software production. For example, the process of developing a whole software system.
2. Need to measure the productivity of staff in order to determine pay settlements for different divisions.
3. Need to measure the quality of the software products.
4. Need to define measurable targets for projects like how much test coverage should be achieved and how reliable the final system should be.
5. Need to measure repeatedly a particular process and resource attribute in order to determine which factors affect cost and productivity.
6. Need to evaluate the efficiency of various software engineering methods and tools.

Engineers:

1. Need to monitor the quality of evolving systems by making process measurements.
2. Need to specify quality and performance requirements in strictly measurable terms, in order that such requirements are testable.
3. Need to measure product and process attributes for the purpose of certification.
4. Need to measure attributes of existing products and current processes to make predictions about future ones.

2.2 Fundamentals of Measurement

A measurement is an objective assignment of a number (or symbol) to an entity to characterize a specific attribute in such a way that captures our intuitive understanding about the attribute. It requires the identification of intuitively understood attributes possessed by clearly defined entities. Measurement can be direct or indirect. Direct measurement of an attribute is a measurement, which does not depend on the measurement of any other attribute, while indirect measurement of an attribute involves the measurement of one or more other attribute [3].

Proper measurement principles have been adhered to software engineering while the term 'metric' has been used in the following distinct ways in the software engineering / metrics literature [3]:

1. A number derived from a product, process or resource. For example the 'metric' *number of function points* or the 'metric' *Lines of Code per programmer month.*
2. A scale of measurement. For example, nominal, ordinal, interval, ratio and absolute.
3. An identifiable attribute. For example, 'metric' portability of programs or the 'metric' coupling in designs.
4. A theoretical or data driven model describing a dependent variable (such as effort) as a function of independent variables (such as program 'size').

It is also worth noting that there have been attempts to distinguish the notions of metric and measure in software in the following way:

1. Metrics numerically characterize "simple" attributes like length, number of decisions, number of operators or number of bugs found, cost and time (for processes) [3]
2. Measures are "functions" of metrics, which can be used to assess or predict more "complex" attributes like cost or quality [3].

2.3 Definitions of Software Metrics

Software metrics deals with the measurement of software product and the processes by which it is developed. The measurement of the software product and process are developed for use in modeling the software development process. Many metrics and models have been proposed to measure and evaluate the software development process, predict faults in software, analyze programs and measure the complexity of software maintenance. Software metrics also can be used to model the development effort and schedule, development cost and maintenance cost. For example, we might want to determine complexity to estimate the effort required for the software development. The definition of software metrics is used to cover all aspects of quantification related to process, resources and product. It may be considered as adequate definition if it is agreed to allow the term 'metrics' to include values obtained in two different ways [6]:

- By estimation – this occurs when the value of a metric is needed at a stage in the development process when it is not available for direct measurement. Predictions or estimates may be based on estimation models or subjective guesses but are derived from knowledge of the product to be developed.

- By measurement – this occurs when the value of a metric can be obtained directly.

A measure is valid if it accurately characterizes the proposed attribute and a prediction system is valid if it makes accurate predictions.

Targets, predictions and actual values are needed in the context of project control. Targets provide the constraints within which a project manager must work. Predictions provide an indicator whether a project is likely to achieve its target while, actual values measure the attributes of the project directly. They also act as the input for identification of targets for future projects and the improvement of estimation methods.

Good metrics is capable in predicting the process, product or resource in the development process. It should be [6]:

1. Simple, precisely definable so that it is clear how the metrics can be evaluated.
2. Valid where the metric should measure what it is intended to measure.
3. Robust where relatively insensitive to insignificantly changes in the process, product or resource.

2.4 Classifications of Software Metrics

Software metrics can be classified into a product, process and resource metrics. Product metrics is a measurement of the software product. It measures the complexity of the software design (such as flow control, depth of nesting or recursion) or the size of the final program (such as the number of lines of code or some count tokens in the program). Process metrics measures of attributes of the software development process and of the development environment. It relates with a time factor, which could be simply a time slice of any software project. It measures a reasonably well defined activities such as constructing a specification document, developing a software system from requirements capture and the overall development time. Resource metrics are items that considered as inputs to software development such as personnel, tools and methods. Interestingly, a product can be a resource for another process. Each software metrics in any classifications can be either internal or external metrics. Internal metrics are those which can be measured in terms of itself (inherent to its entity) and external metrics are measured with respect to how the properties of entities relates to its environment.

Another way in which software metrics can be categorized are as development productivity, project management, quality and software development. Productivity metrics controls the cost and effort needed to develop the software development. Project management metrics will track the progress of a project. While quality metrics verify the correctness, complexity, coupling of modules and variables, reusability, modifiability, testability, maintainability and performance.

Software development metrics determines the metrics required according to the life-cycle stage. Table 2.1 shows the metrics according to the life-cycle development phase [3].

Table 2.1

Usefulness of Metrics

Stages	Resources	Usefulness Of Metrics At The Current Stage
Requirement analysis	Requirement specification	Cost estimation Design Testing Maintenance
Design	Design specification	Coding guidance Comparison of different designs
Implementation	Source/codes Comments Documentation	Testing Maintenance
Testing	Test data Test routines	Testing Maintenance
Maintenance	All above resources and history	Maintenance

2.5 The Scope of Software Metrics

Software metrics embraces many activities, which involve some degree on software measurement. The scope of the software metrics can be categorized into:

1. **Productivity measures and models.**
   It measures the productivity of personnel during software processes. The measurement is based on the size of output divided by effort [3].

2. **Data collection.**
   It measures the data collection whether the data is valid, accurate and precise. It is also described a general data collection methodology for reliability assessment [3].

Software development metrics determines the metrics required according to the life-cycle stage. Table 2.1 shows the metrics according to the life-cycle development phase [3].

Table 2.1

Usefulness of Metrics

Stages	Resources	Usefulness Of Metrics At The Current Stage
Requirement analysis	Requirement specification	Cost estimation Design Testing Maintenance
Design	Design specification	Coding guidance Comparison of different designs
Implementation	Source/codes Comments Documentation	Testing Maintenance
Testing	Test data Test routines	Testing Maintenance
Maintenance	All above resources and history	Maintenance

2.5 The Scope of Software Metrics

Software metrics embraces many activities, which involve some degree on software measurement. The scope of the software metrics can be categorized into:

1. **Productivity measures and models.**

   It measures the productivity of personnel during software processes. The measurement is based on the size of output divided by effort [3].

2. **Data collection.**

   It measures the data collection whether the data is valid, accurate and precise. It is also described a general data collection methodology for reliability assessment [3].

3. **Reliability models.**

   The work on measuring and predicting software reliability can be seen as a rigorous and successful example of a detailed study of specific software product attributes [3].

4. **Performance evaluation and models.**

   It concerns with the specific product attribute such as the system performance aspects like response time and the efficiency of algorithms [3].

5. **Structural and complexity measures.**

   The structural and complexity metrics measures the structural attributes of representations of the software. Two classical examples of this approach are the Halstead and McCabe which measure the complexity and the structural of source code [3].

6. **Cost and effort estimation models and measures.**

   It measures the project costs at the early stage in the software development process. There are numerous models for cost and effort estimation that have been proposed such as COCOMO model, Putnam's SLIM model and Albrecht's function points model [3].

2.6 The Use of Software Metrics

Generally software metrics are used in two important ways:

- Software metrics as control metrics in the control management of the development process.
- Software metrics as predictor metrics of either product qualities or control metrics.

Control metrics is used in project control of any software development process. Any industrial production or manufacturing activity would be managed and controlled by control metrics. The type of metrics that is widely used for project control is resource-related metrics such as effort,

development schedule and machine usage for particular activities. This metrics incorporated into a management planning and monitoring activity where estimates of the effort, development schedule and machine usage are made as part of the project plans and actual values are used to monitor progress against those plans. Other metrics used in project control are those used to estimate task completion. For example, they compare the estimate of the "size" of the task in terms of its expected output is compared with the amount of the task of its actual output, which has been completed at a particular point of time. Defect-related metrics is another control metrics that is often used in project control. Defect-related metrics is the discovery and elimination of defects. It is the major cost of non-conformance. To control the non-conformance, it is necessary to record information about the defects and the costs associated with their discovery and removal. Defect rate can be defined as estimated defect rates based on data from past projects. It is also used to monitor the defect rates in order to ensure the activities and the product is behaving as expected.

Predictor metrics is used to provide estimate of control metrics. It is used to estimate the final product qualities and also used as input parameters to equation, which predicts the product characteristics such as "reliability", "maintainability" or predicts the control metrics such as " the number of faults detected during testing phase ". Most predictive equations are usually derived empirically using statistical techniques.

2.7 Framework for Prediction Measures

For the purpose of prediction, a model and a prediction system need to be defined. The prediction system, associated with a model consists of a number of prediction procedures for determining unknown parameters and interpreting results. A process for making predictions must have three activities [7]:

- Selecting and modelling the measures to be predicted
- Making predictions
- Evaluating the accuracy of the predictions

The first step in the prediction process is to select the measures to be predicted for a project and the methods by which to predict them. The measurement prediction is necessary to reflect the project's target. In this case, a measure is a numeric representation of a specific attribute of an entity. Measures for system development represent attributes of process inputs, outputs or the process itself. For example, in measuring the attribute of effort, a count of delivered lines of code is one measure of the length of an input to a system development process and the number of hours spent on developing the code is a measure of effort for the process [7]. The method by which the measure is predicted must also be selected. A regression model, an algorithmic model, neural network or fuzzy logic model could be used in predicting the effort. Effort can be measured by using fuzzy logic approach in order to give a good prediction. This is done by specifying the linguistic variables involved in the effort prediction. Experience, size of the project and use of tools are the examples of linguistic variables. Linguistic variables are the term whose values are words in natural language. For example, the linguistic values for size of the project are very small, small, medium, high and very high.

In the second step of the prediction process, the predictions are made. As data become available, the predictions are analyzed to assess whether they are being met. The results of this analysis are fed back to those making the predictions. This feedback gives the developers an opportunity to take corrective action if appropriate [7].

The final step involves evaluating the accuracy of the predictions. This process is needed to improve the accuracy of predictions. It is necessary to learn from experience when dealing with this step [7]. The reasons for inaccurate predictions need to be understood and the findings must be recorded. These findings should be accompanied by suggestions for improving the accuracy of future predictions [7].

Predictions are usually based on indirect measures because the attribute of interest is not directly measurable at the time the prediction is made. For example, if the effort required for system development process is predicted from the length of the software to be produced, a prediction of the length of the software is needed too. Effort is the attribute of processes and length is the

attribute of the product. Effort is measured using ratio scale [7]. This is usually recorded manually and often leads to inaccuracies of measurement.

2.8 Summary

Software metrics are measurements concerning either the software being developed (the product) or the manner in which it is being developed (the process). The metrics used in project may be grouped into different categories, namely:

- Metrics related to resources such as effort and staffing level through development.
- Metrics relating to specification and design documents such as size and structure of a program.
- Metrics relating to structural attributes of representations of the software.

To assist software project, it is necessary to identify an appropriate metrics for each phase in the project life cycle. The prediction process presented in this chapter is the basis for analyzing and interpreting the software cost estimation. Software cost estimation will be discussed in the next chapter. The prediction process highlights the importance of measurement within the context of system development because measurement provides the data on which predictions are based.

CHAPTER 3

SOFTWARE COST ESTIMATION

Accurate software cost estimation is critical in software development process. Underestimate cost estimation will have many undesirable effects such as over or under allocation of development resources and personnel. As a result, many models have been proposed to avoid these problems. In this chapter we introduce the software cost estimation and describes the problems associated with its process. The major software cost estimation models such as COCOMO, Function Points and SLIM will be discussed, together with their limitations. Finally, we will review fuzzy logic as an alternative technique for cost estimation.

3.1 Software Cost Estimation: An Overview

Software cost estimation is commonly known, as making estimates of the effort required to complete a project for a system development activity. Generally, software cost estimation does not estimate cost in cash terms but measures the effort, duration and manpower loading. Manpower loading is the number of engineering and management personnel allocated to the project as a function of time. Duration is the amount of time required to completing the project. Effort is measured in man-months of the technical staff required to complete a project. A technical staff normally includes programmers, analyst, software engineers and management directly associated with the project.

The purposes of software cost estimation are to prepare project planning and assessing the feasibility of system development. Planning plays an important role to determine the success of the management of system development activities. Inaccurate estimates of software cost during project planning can lead to underestimate of effort, overrun cost and delivery time. An

overestimate of effort may also adversely affect the project where it leads to overstaffing. A critical problem faced by all project managers is the inaccuracy of software cost estimation. A decision by management whether to proceed with the project depends on the estimated cost. Even the project done in-house or contracted put is likely to be affected by cost and effort estimates.

Cost estimation requires good measurement of quantitative data. Basically, cost estimation is based on the size of the estimation and the relationship between the various parameters of the software development such as project duration, total effort, documentation, code delivering, staff size and computer cost. Software cost estimation is a continuing activity, which starts at the proposal stage and continues throughout the project life cycle. Preliminary estimation is required to determine whether a project is feasible in terms of cost-benefit analysis. Usually, a preliminary estimate is difficult and least accurate because very little detail is known about the project. Thus, in order to determine the preliminary estimates, there are many unknown quantities that must be estimated. These are associated with the resources availability at the required time, project development cost, project development schedule and the size of the project development team.

It is desirable to know how much is the cost to develop the software and how much time development will be needed. These estimates are needed before development is initiated. Let us consider several limitations of cost estimations. The accuracy of the cost estimates will be determined when the product is delivered. All the data about the project and the resources spent on it, is fully known by then. There is uncertainty in the specifications of the system where it represents a range of possible final products. Here, the cost estimation cannot be accurate. If the system is specified more accurately, the uncertainties can be reduced and more accurate cost estimates can be made. Figure 3.1 [5] shows the fundamental limitation of software cost estimation where the accuracy within which software cost estimates can be made, as a function of the software life-cycle phase. This level of uncertainty is illustrated with the respect to a human-machine interface component of the software. Note that, it does not show the strategies that can provide the estimate with that accuracy.

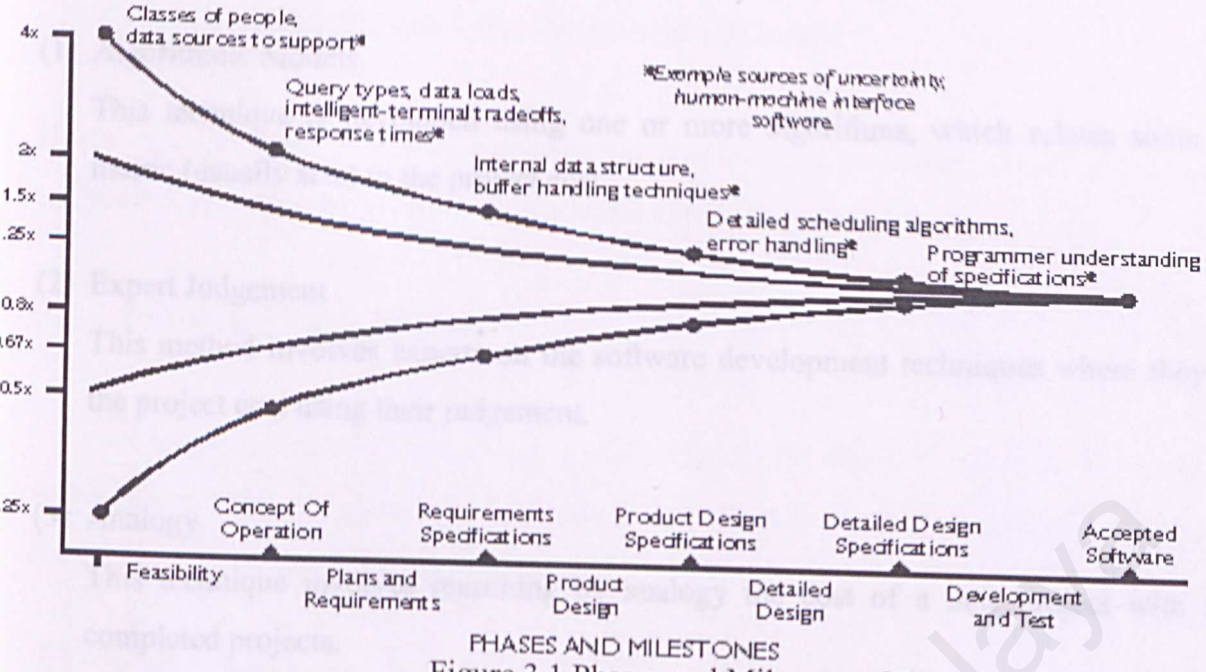

Figure 3.1 Phases and Milestones [5]

3.2 Software Cost Estimation Techniques

Boehm [5] describe seven techniques of software cost estimation. Table 3.1 summarizes the strengths and difficulties of these techniques.

Table 3.1

Strengths and Weaknesses Of Software Cost-Estimation Methods

Method	Strengths	Weaknesses
Algorithmic Models	• Objective, repeatable, analyzable formula • Efficient, good for sensitivity analysis • Objectively calibrated to experience	• Subjective input • Assessment of exceptional circumstances • Calibrated to past, not future
Expert Judgement	• Assessment of representatives, interactions, exceptional circumstances	• No better than participants • Biases, incomplete recall
Analogy	• Based on representatives, experience	• Representatives of experience
Parkinson	• Correlates with some experience	• Reinforces poor practice
Price-To-Win	• Often gets the contract	• Generally produces large overruns
Top-Down	• System level focus • Efficient	• Less detailed basis • Less stable
Bottom-Up	• More detailed basis	• Requires more effort

(1) Algorithmic Models.

This technique is developed using one or more algorithms, which relates some software metric (usually size) to the project cost.

(2) Expert Judgement

This method involves experts on the software development techniques where they estimate the project cost using their judgement.

(3) Analogy

This technique involves reasoning by analogy the cost of a new project with the other completed projects.

(4) Parkinson

Parkinson principle states that the cost is determined to the available resources rather than by objective assessment. If the software has to be delivered in 12 months and 5 people are available then the effort required is estimated to be 60 person-months.

(5) Price-To-Win

The cost estimated equated to the price the customer has spend on the project. The estimated effort depends on the customer's budget and not on the software functionality.

(6) Top-Down

An overall cost estimation for the project is established by considering the overall functionality of the properties of the software product.

(7) Bottom-Up

The cost of each component is separately estimated and the results of all these costs are added to produce an estimate for a final cost estimate.

3.3 Traditional Software Cost Estimation Models

Below are the descriptions of the software cost estimation model, which are based on the software cost estimation techniques, discussed earlier in section 3.2.

3.3.1 Putnam SLIM Model

Putnam developed a constraint model called SLIM which is based on his analysis of the software life cycle development. It proposed that effort for software projects is distributed to a collection of Rayleigh curves shown in figure 3.2 [3].

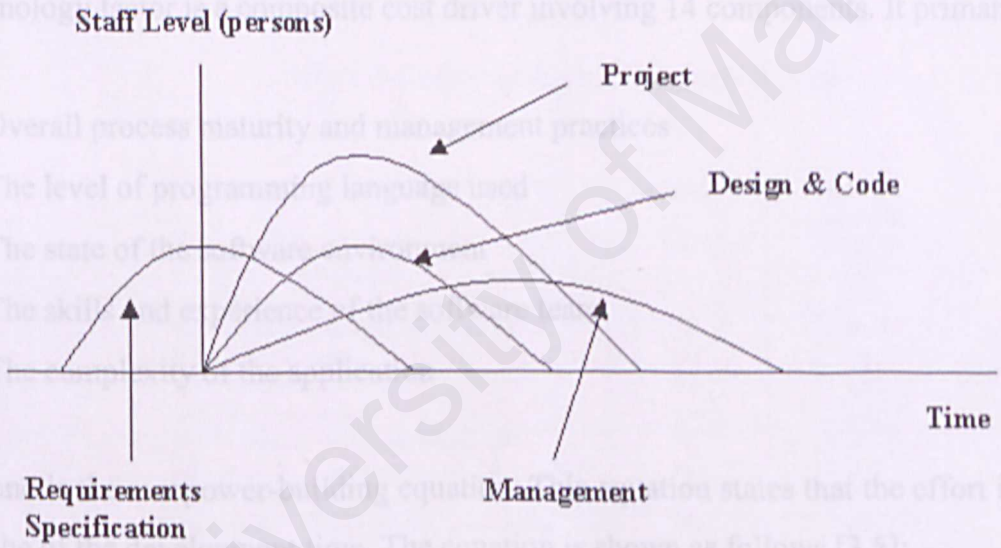

Figure 3.2 Collection of Rayleigh Curves for SLIM Model

Putnam's analysis shows that staffing rises smoothly during the project and then drops during testing. The SLIM model is expressed in two equations describing the relation between the development effort and the schedule. The first equation called the software equation states that the development effort is proportional to the cube of the size.

The equation is expressed as follows [3,5]:

$$\text{Size} = C_k K^{1/3} t^{4/3}_d$$

where size is measured as lines of code to several variables, K is the total life cycle effort (in working years), t is the development time (in years) and C_k is a technology constant. The values of C_k can range between 610 and 57, 314. Putnam has recommended 1500 for a real-time embedded software project, 4894 for a batch development environment and 10 040 for a well-supported and well-organized environment.

The technology factor is a composite cost driver involving 14 components. It primarily reflects:

- Overall process maturity and management practices
- The level of programming language used
- The state of the software environment
- The skills and experience of the software team
- The complexity of the application

The second is the manpower-building equation. This equation states that the effort is proportional to the cube of the development time. The equation is shown as follows [3,5]:

$$D_0 = K / t^3_d$$

where D_0 is a constant called manpower acceleration which takes specific value for a particular type of a project.

The Putnam's SLIM model has several limitations. This model did not show the evidence of the relationship between 'difficulty' ($K / (t_d)^2$) and productivity that Putnam used to construct the

software equation. The design and code start from zero staffing level. Parr [5] suggested that it should not start from zero staffing level and proposed an adjustment to the curve.

3.3.2 The Doty Model

The Doty model that was published in 1977 was used to estimate the manpower, cost and development time for software project. The first step of using this model is to determine the 'effort adjustment factor' (EAF). The EAF is fourteen products effort multiplier. Table 3.2 [16,17] shows EAF.

Table 3.2

Effort Adjustment Factor For Doty Model

Factor	Applicable	Effort Multiplier
Special display	No	1.00
Detailed definitions of operational requirements	Yes	1.00
Change to operational requirements	No	1.00
Real-time operation	No	1.00
CPU memory constraint	Yes	1.43
CPU time constraint	Yes	1.33
First software development on CPU	No	1.00
Concurrent development on ADP hardware	No	1.00
Timeshare versus batch processing in development	No	1.00
Developer using computer at another facility	No	1.00
Development at operational site	No	1.00
Development computer different from target computer	Yes	1.25
Development at more than one site	No	1.00
Programmer access to computer	Unlimited	0.90
	EAF	**2.14**

Each effort multiplier has two values depending on whether the particular factor is relevant. When the EAF is known, the overall effort will be determine by using a simple estimated effort Doty algorithm. The equation is expressed as follows [16,17]:

$$MM_{est} = 2.06 * (KDSI)^{1.047} * EAF$$

where MM_{est} is man-month estimate.

One primary advantage of Doty model is that it is an objective, repeatable and removes the need for an 'expert' opinion (where there is no previous experience of the type of project being developed, this advantage is considerable). The estimating process is also relatively rapid [16,17].

3.3.3 The PRICE-S Model

The PRICE-S model [16] is a commercially macro cost-estimation model which makes a top-down estimate of the resources required in software development. This model computes cost, as well as the schedule, size, type and difficulty of the proposed project. PRICE-S uses a two-parameter beta distribution rather than a Rayleigh curve to calculate development effort distribution. The model first computes the empirical factors from historical cost databases and then uses these factors in the cost estimation of new projects.

3.3.4 COCOMO

COCOMO (Constructive Cost Model) was developed by Boehm. This model provides specific formulas for estimating the effort, development time schedule, effort by phase and activity and maintenance effort. COCOMO is relatively straightforward model (based on inputs) relating to the size of the system and a number of cost drivers that affect productivity.

COCOMO is based on two equations which are shown as follows [3,4,5,6,8,9]:

$$MM = a * KDSI * b$$

$$TDEV = c * MM * d$$

where

MM is the effort in person-months

KDSI is the number of thousand-delivered source instructions

TDEV is the development time

Coefficients a,b,c and d are dependent upon the 'mode' of development which are shown in Table 3.3 [5].

Table 3.3

Value Of The Coefficient

Development Mode	a	B	c	D
Organic	3.2	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

There are three types of models for COCOMO: a Basic model applied early in the project, an Intermediate model that is applied after requirements are specified and a Advanced model that is applied after design is complete. In the Intermediate COCOMO, there are two stages that need to perform. The first stage involves deciding the projects's development mode. Table 3.4 [5] is used to determine the project's development mode.

Table 3.4

COCOMO Software Development Modes

Feature	Organic	Semidetached	Embedded
Organizational understanding of product objectives	Through	Considerable	General
Experience in working with related software systems	Extensive	Considerable	Moderate
Product size range	< 50 KDSI	< 300 KDSI	All sizes

The second stage is to determine the project's fifteen cost drivers. Each cost driver has a rating scale and a set of effort multipliers, which indicate the rating level for the project's attribute. The effort adjustment factor (EAF) is calculated using 15 cost drivers. The cost drivers are grouped into four categories: product, computer, personnel and project. Each cost driver is rated on a six-point ordinal scale ranging from low to high. Based on the rating, these cost drivers attributes and their corresponding effort multipliers are determined as shown in figure 3.5 [5].

Table 3.5

COCOMO Cost Drivers And Effort Multipliers

Cost Driver	Very Low	Low	Nominal	High	Very High	Extra High
Product Attributes						
RELY	0.75	0.88	1.00	1.15	1.40	
DATA		0.94	1.00	1.08	1.16	
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer Attributes						
TIME			1.00	1.11	1.30	1.66
STOR			1.00	1.06	1.21	1.56
VIRT		0.87	1.00	1.15	1.30	
TURN		0.87	1.00	1.07	1.15	
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	
AEXP	1.29	1.13	1.00	0.91	0.82	
PCAP	1.42	1.17	1.00	0.86	0.70	
VEXP	1.21	1.10	1.00	0.90		
LEXP	1.14	1.07	1.00	0.95		
Project Attributes						
MODP	1.24	1.10	1.00	0.91	0.82	
TOOL	1.24	1.10	1.00	0.91	0.83	
SCED	1.23	1.08	1.00	1.04	1.10	

RELY - Required software reliability
DATA - Data base size
CPLX - Product complexity
TIME - Execution time constraint
STOR - Main storage constraint
VIRT - Virtual machine volatility
TURN - Computer turnaround time
ACAP - Analyst capability
AEXP - Applications experience
PCAP - Programmer capability
VEXP - Virtual machine experience
LEXP - Programming language experience
MODP - Use of modern programming practices
TOOL - Use of software tools
SCED - Required development schedule

The results of applying table 3.5 are shown in table 3.6 [5].

Table 3.6

Results Of COCOMO Cost Driver Ratings

Cost Driver	Situation	Rating	Effort Multiplier
RELY	Serious financial consequences of software faults	High	1.15
DATA	20,000 bytes	Low	0.94
CPLX	Communications processing	Very high	1.30
TIME	Will use 70% of available time	High	1.11
STOR	45K of 64K store	High	1.06
VIRT	Based on commercial microprocessor hardware	Nominal	1.00
TURN	Two-hour average turnaround time	Nominal	1.00
ACAP	Good senior analyst	High	0.86
AEXP	Three years	Nominal	1.00
PCAP	Good senior programmers	High	0.86
VEXP	Six months	Low	1.10
LEXP	Twelve months	Nominal	1.00
MODP	Most techniques in use over one year	High	0.91
TOOL	At basic minicomputer tool level	Low	1.10
SCED	Nine months	Nominal	1.00
	EFFORT ADJUSTMENT		1.35

The full algorithm for organic-mode intermediate COCOMO is expressed as follows [5]:

$$MM_{est} = 2.4 * (KDSI)^{1.05} * EAF$$

In the COCOMO advanced model the project is divided into four phases: Product Design, Detailed Design, Coding/Unit Testing and Integration/Test. This model computes effort as a function of program size and a set of cost drivers weighted according to the four phases. The fifteen cost drivers are estimated and applied to each phase separately rather than as a whole.

COCOMO also provides equations for nominal development time in months. For example, the development schedule equations are varied by mode [5]:

$$T = 2.5 * E^{0.38} \quad \textbf{(Organic Mode)}$$

$$T = 2.5 * E^{0.35} \quad \textbf{(Semi-detached Mode)}$$

$$T = 2.5 * E^{0.38} \quad \textbf{(Embedded Mode)}$$

The advantages of COCOMO are very similar to Doty model where the COCOMO model producing a prediction relatively rapid. There is a greater level values in the EAF rather than the two values (yes or no) with the Doty model. In the COCOMO there is a choice of five cost driver ratings ranging from 'very low' to 'extra high'.

There are many disadvantages of the COCOMO model. A prediction of the system size (KDSI) must be made before the effort can be calculated thus, probability to get an accurate result is very low. Secondly, the selection of the cost driver rating tends to be somewhat subjective and more complex than the Doty model. The cost driver also tends to be difficult in selecting between 'very high' and 'extra high', yet this could make a difference in the overall estimate of 30%. Another problem is that this model involves too many parameters to be estimated. A simpler model could be constructed on three or four parameters by combining some of the attribute into a single cost driver.

3.3.5 Function Points

Function Points [9,18,19,20,21,22] was developed by Allan Albrecht in 1979, as an alternative to estimating SLOC (source lines of code). Function Points are based on characteristics of the project that are at a higher descriptive level than SLOC such as the number of input transaction types and number of reports. This approach characterizes the amount of user functionality provided by the product based on the following characteristics [9,21,22]:

- External inputs
- External outputs
- Inquiries
- Internal files
- External or interface files

Each of the categories is counted individually and then weighted by numbers reflecting the relative value of the function to the user. After identifying and classifying the user functions, the Function Points are adjusted to reflect function complexity and application complexity. Function Points are counted by itemizing and weighting functionality by user and are calculated using the following formula [9,21,22]:

$$\mathbf{FP = UC * (0.65 + 0.01 * TCF)}$$

where

UC is the Unadjusted Count

TCF is the Technical Complexity Factor.

The unadjusted count is defined as follows [9,21,22]:

$$\mathbf{UC = aI + bO + cE + dL + eF}$$

where

I is the number of input types

O is the number of output types

E is the number of inquiry types

L is the number of logical internal files

F is the number of interfaces

While a,b,c,d,e > 0 are the weighting factors namely, determined in the table 3.7 [9,21,22].

Table 3.7

Weights For The Unadjusted Function Count (UC)

Type	Simple	Average	Complete
I	3	4	6
O	4	5	7
E	3	4	6
L	7	10	15
F	5	7	10

The technical complexity factor (TCF) is defined as follows [9]:

$$\mathbf{TCF = 0.65 + 0.01\ \Sigma\ F_i}$$

where there are fourteen factors and weights from zero to five. The fourteen factors are shown in table 3.8 [9]:

Table 3.8

Fourteen Technical Complexity Factor

Data Communication	4
Distributed Function	3
Performance	5
Heavily Used System	4
Transaction Rate	5
On-Line-Data Entry	5
End-User Efficiency	1
On-line Update	4
Complex Processing	2
Reusability	2
Installation Ease	1
Operational Ease	4
Multiple Sites	0
Facilitate Change	3

One promising use of Function Points is in measuring the size. Most of the effort estimation models such as COCOMO use software size as a major cost driver. Unfortunately, the size of the product is usually not known in the early development stage. Function Points offers several significant advantages. It is possible to estimate the size in the early development of life cycle using Function Points [9]. This can be an important advantage for anyone who is trying to estimate the level of effort to be required on a software-development project [9]. They can also avoid the effects of language and other implementation differences [9].

Function Points also have its limitations. The measurement of TCF does not assume an extensive structure. The technical complexity factor is misleading because the numbers may suggest a ratio scale. These fourteen factors confuse the user where each factor has to be decided in its weights factors intuitively [9]. If the factors are considered as ranking factors in the sense that, the more complicated the system is the higher the number, then another effect where setting two other factors from two to zero can be equalized by another factor from 0 to 4.

Another limitations of Function Points are that it requires a significant amount of human labor and judgement. The importance of judgement raises questions about the reliability of the function point measurement. It is unclear whether a Function Points count might not vary between analysts [9]. As such, constant Function Point count could not be guaranteed.

3.3.6 Regression-based Models

Regression technique is the earliest technique used in building the cost models. Regression analysis is commonly preceded by the creation of two-dimensional scatter plots and exploratory correlation analysis in order to first intuitively as well as quantitatively determine the potential relationships that may exist in the data [3]. By collecting data from the past projects and examining relationships among the attribute, the software engineer can hypothesize some factors and derive the basic equation.

Figure 3.3 [3] shows how regression equation might be derived. Each data point represents a project. The regression has been performed using the logarithm of project effort on the y-axis and the logarithm of project size on the x-axis. Transforming the linear equation is defined as follows [2]:

$$\text{Log } E = \log a + b \log S$$

From the log domain to the real domain yields an exponential relationship of the form [3]:

$$E = aS^b$$

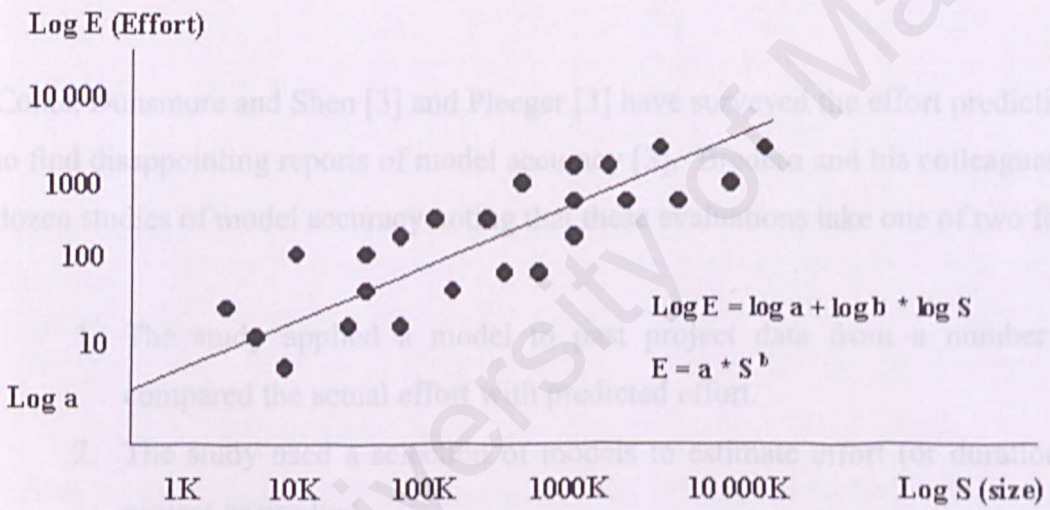

Figure 3.3 Regression Line Generated For Cost Model

If size were the accurate predictor for the effort, then every point in the graph would lie on the line of the equation with a residual error of 0 [3]. However, data is often highly skewed. For instance, module size data tends to be significantly skewed to the right due to the influence of a few very large modules [3]. The next step in regression-based model is to identify the factors that cause variation between predicted and actual effort. 80% of the variation in the required effort for similar-sized projects is explained by the experience of the programming team [3]. A factor analysis can help in identifying the additional parameters and adds these as cost drivers and assign weighting factors to the model. For example, if developers have 8 to 10 years of

experience, then a weight may define for "low" experience (under 8 years) as 1.3, "medium" (8 to 10 years) as 1.0 and "high' (more than 10 years) as 0.7. The weights should be based on empirical data, not on expert judgement. These weights are applied to the right-hand side of the effort equation which yielding a model of the form [3]

$$E = (aS^b)F$$

where F is the effort adjustment factor, computed as the product of the cost driver value. This computation of F is valid only when the individual adjustment factors are independent [3].

3.3 Problems With Existing Modeling Methods

Conte, Dunsmore and Shen [3] and Pleeger [3] have surveyed the effort prediction literature, only to find disappointing reports of model accuracy [3]. Bredero and his colleagues cited almost two dozen studies of model accuracy noting that these evaluations take one of two forms [3]:

1. The study applied a model to past project data from a number of projects, and compared the actual effort with predicted effort.
2. The study used a selection of models to estimate effort (or duration) for a particular project or product.

From the first case, the actual values are different form the predicted values, even when all the input values are known. In the second case, the different models give different results for the same inputs. All of the studies above show a clear picture of model insufficiency. There are several reasons why the existing models are insufficient and these are discussed in the below.

3.4.1 Model Structure

Most researchers agree that the size of the product is the key to determine the effort required to build the product. The relationship between effort and size are modeled as a function of size with

an exponent b and a multiplicative term a. Most models which the effort is proportional to size, include an adjustment of scale (b slightly greater than 1) so that larger projects are less productive than small ones [3]. Many model builders feel that the duration of a project is proportional to the effort. Table 3.9 [3] shows the exponential term in effort-duration models.

Table 3.9

Exponential Term in Effort-Duration Models

Dataset	B	Se(b)	>0	=0.333	Projects
Bailey-Basili	0.167	0.076	Yes	No	19
Belady-Lehman	0.420	0.046	Yes	Yes	33
Yourdon	0.252	0.080	Yes	Yes	17
Wingfield	1.016	0.625	No	-	15
Kemerer	0.315	0.158	No	-	15
Boehm					
-All	0.375	0.031	Yes	Yes	63
-Organic	0.438	0.074	Yes	Yes	23
-Semi-detached	0.458	0.054	Yes	No	12
-Embedded	0.400	0.057	Yes	Yes	28
Kitchenham-Taylor	0.262	0.262	Yes	Yes	33

Putnam's model implies that decreasing duration will increase effort and increasing duration with decrease the effort. Boehm's model on the other hand, assumes that increasing or decreasing in the duration of development time will increase the project effort. The empirical results by Jeffery have added more confusion on this matter. He calculated the extent of schedule compression and effort compression for a number of projects and then constructed a scatter plot comparing the two. The resulting graph included schedule-compressed projects that were also effort-compressed, contradicting both the Boehm and Putnam models [3]. Most models perform poorly due to the input parameters, which are, difficult to access accurately at the start of the project.

3.4.2 Overlay Complex Models

Some of the models include the adjustment factors such as COCOMO's cost drivers and SLIM's technology factors to improve the accuracy of estimates. However these adjustment factors have fallen short of its promise:

1. Application of the COCOMO cost drivers does not always improve the accuracy of estimates [3].
2. It is not easy to obtain an accurate estimate of the technology factor and SLIM estimates are extremely sensitive to the technology factor [3].
3. The cost drivers are not independent but they are treated as if they were by COCOMO and similar models. This situation leads to inappropriate multiplication of cost drivers and equations that are likely to be unstable when applied to data sets other than the ones from which they were derived [3].
4. Cost driver values are usually based on subjective assessments of the influence of some factors (such as personnel experience) on overall project effort. It is difficult to ensure that different people assess the factors consistently and the factors are assessed by the model user in the way that was intended by the model producer [3].
5. Current models include many adjustment factors, allowing the estimator to cope with many different types of project. However, the projects undertaken by a single development group are often quite similar and only a few factors need to be considered [3].

3.4.3 Product Size Estimation

Most models require an estimate of product size in order to estimate the effort required to complete a project. Size is the most important single cost driver and a good effort estimate depends on good size estimation. Obtaining good size estimation is difficult and often led to a wide margin of uncertainty. COCOMO and SLIM models require size in lines of code (LOC). Evidence has shown that estimates of lines of code can be inaccurate [3].

Size can be estimated in different views of the structure of the software. Some estimate the size of a system as a whole. While others break a system up into components where the size are estimated separately.

3.4.4 Providing Exact Values For Inputs

Most models require exact values for input in estimating. Major problems that exist in such model is that the difficulty of project manager to provide accurate or exact values for input. There are studies of model accuracy stated that these evaluations take one of two forms [1].

1. The study applied a model to past project data from a number of projects and compared to the actual effort with predicted effort. The actual values are different from the predicted values even when all the input values are known [1].

2. The study needs a selection of models to estimate effort or duration for a particular project or product. The different models give very different results for the same input [1].

The output from such models may lead to overconfidence in accuracy and precision of the result. For example, the dependent variable is predicted as 7121.6 person hours, there is a risk that would lead to development time being wasted in overestimate and requirements remaining unfulfilled or the project going over schedule [1]. Therefore, it pictures the insufficiency of the existing models.

3.5 Alternative Measure for Cost Estimation

Most software effort modeling techniques relied on algorithmic methods. In this method, the researchers have to examine data from past projects and generate equations from them to predict effort and cost for future projects. Since, this method relies on empirical data to be mapped with the equation, researchers are looking for other technique to produce good effort estimation. One alternative is the fuzzy logic.

The use of fuzzy logic for software metrics cost estimation model [1,19,29,30,31,32,33] seems to be appropriate for using the existing expert knowledge from project managers. They use some form of expert judgement as part of the project planning. A project manager may specify a project as small, medium or large. This can be represented as fuzzy variables and fuzzy rules to derive some prediction for the output. This approach is demonstrated in figure 3.4 [29,31,32].

In this example, numerical values are provided for data model size (30), number of screens (26) and process model size (74). These values are plotted on the membership functions with the height of intersection with the membership curve indicating the degree to which the value belongs to the respective label. Figure 3.4 [29,31,32] shows the values are plotted on their respected functions.

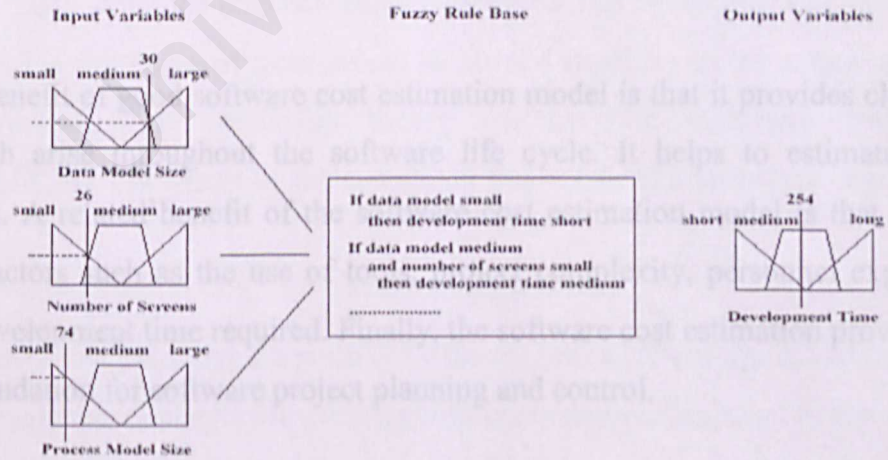

Figure 3.4 A Fuzzy System for Duration Estimation

The data model size function indicates an intersection between medium and large. Therefore, the truth-value for both levels of logic is to a degree of 0.5. The number of screens function indicates an intersection between small and medium, so the truth-value for both levels of logic is to a degree of 0.5. The final function, which is the process model size only, intersects inside the small section and is given truth-value of 0.8.

The truth-values are then used in the fuzzy rule. The degree of memberships in each category will determine how much weight is needed to be given to the fuzzy rule. The consequences of each rule are then combined to produce a single output value indicated as 254. This process is called defuzzification.

3.6 Summary

In summary, although progress has been made in software cost estimation, the biggest difficulty in using today's software cost models is in the determination of the estimates. Every model requires an estimate of the number of source or object instruction to be developed and this can be extremely difficult to determine. The primary implication of this situation is that there is no magic formula that will provide an accurate estimate of the software sizing.

The major benefit of good software cost estimation model is that it provides clear and consistent issues, which arise throughout the software life cycle. It helps to estimate the cost of the development. A related benefit of the software cost estimation model is that it considers other influential factors such as the use of tools, project complexity, personnel experience, software reuse and development time required. Finally, the software cost estimation provides an absolutely essential foundation for software project planning and control.

CHAPTER 4

FUZZY LOGIC

In this chapter, we will look into the ideas of fuzzy logic in handling real world uncertainty. We review some of the essential characteristics of fuzzy sets, its properties and operations. The types of inference method and defuzzification process are addressed at the end of this chapter.

4.1 Overview of Fuzzy Logic

In the real world, information is often ambiguous or imprecise. Statement like the weather is warm and the program is very complex, are difficult to interpret. In this case, human reasoning is needed in order to arrive at particular conclusions. Computer programs with fuzzy logic are useful when dealing with the impreciseness of information. In other words, fuzzy logic is an organised method for dealing with imprecise data.

Lukasiewiez [10,11,12,34,35] (the inventor of reverse Polish notation) has proposed fuzzy systems in the 1920s. He studied the mathematical representation of fuzzy terms such as "large", "warm" and "fast". From his understanding, these terms can be mapped into the two-valued [0,1]. He also extended the range of truth-values to all real numbers in the range of 0 to 1 and used these to represent the possibility theory whether the statement was true or false [10,11,12,13,34,35]. For example, the possibility that a size of a project is large might be set to a value of 0.8.

In 1965, Zadeh have extended the possibility theory into a formal system of mathematical logic. He proposed valuable concepts for working with fuzzy natural language terms. This new logic for representing and manipulating fuzzy terms was called **fuzzy logic** [13,35]. Generally, fuzzy logic is a branch of logic that uses degrees of membership in sets rather than a strict true or false membership. Fuzzy logic is basically a multivalued logic that allows intermediate values to be

defined between conventional evaluations like yes or no, true or false etc [13,23,24,25,26,35]. The manipulation terms like very large or very small can simultaneously be seen to belong partially to two or more different contradictory sets of values [13,23,24,26,35]. It is a superset of conventional (Boolean) logic that has been extended to handle the concept of partial truth – truth values between "completely true" and "completely false" [13,34,35].

4.2 Fuzzy Sets

In mathematics, a set is simply a collection of objects. The objects can either belong to the sets or do not belong to the set. It is similar to the logic in statement which can either be true or false. A fuzzy set is one to which objects can belong to different degree of membership. This theory is differ from the conventional classical set theory from one important concept: it allows each element of a given set to belong to that set to some degree which means each element either fully belongs to the set or is completely excluded from the set [24,27].

Suppose X = {x} is a universe of discourse, a fuzzy set A in X is defined as a set of ordered pairs:

$$\{(x, \mu_A(x))\}$$

where $x \in X$ and $\mu_A : X \rightarrow [0,1]$ is the membership function of A;

$\mu_A(x) \in [0,1]$ is the grade of membership of x in A, from 0 for completely not belong to the set to 1 for completely belong to the set. [22,23,27,34].

A fuzzy set in a finite universe of discourse is conveniently defined as follows:

$$A = \mu_A(x_1)/x_1 + \ldots + \mu_A(x_n)/x_n,$$

where "$\mu_A(x_i)/x_i$ " (called a singleton) is a pair "grade of membership-element" and "+" is meant in the set-theoretic sense.

For example, if X = {1,2,…,10}, then a fuzzy set "large number" may be given as A = "large number" = 0.2/6 + 0.5/7 + 0.8/8 + 1/9 + 1/10 to be meant as : 9 and 10 are "large numbers", 8 is a large number to a degree of 0.8, 7 is a large number to a degree of 0.5 and 6 is a large number to a degree of 0.2. It is convenient to use a linear representation of the membership function of a fuzzy set as shown in figure 4.1 where only two values are needed, $\overline{A}$ and A [23].

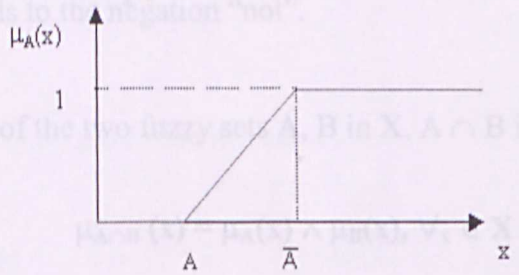

Figure 4.1 Membership function of a fuzzy set

4.2.1 Basic Properties of Fuzzy Sets

A fuzzy set A in X is empty, $A = \varnothing$, if and only if $\mu_A(x) = 0, \forall_x \in X$ [23]. Two fuzzy sets A, B in X are equal,

$$A = B, \text{ if } \mu_A(x) = \mu_B(x), \forall_x \in X.$$

A fuzzy set A in X is included in (is a subset of) a fuzzy set B in X, $A \subseteq B$, if and only if

$$\mu_A(x) \leq \mu_B(x), \forall_x \in X.$$

4.2.2 Basic Operations on Fuzzy Sets

The basic operations on fuzzy sets are the complement, intersection and union, as in the conventional set theory. The complement of a fuzzy set A in X, ¬A, is defined as follows:

$$\mu_{\neg A}(x) = 1 - \mu_A(x), \forall_x \in X$$

and it corresponds to the negation "not".

The intersection of the two fuzzy sets A, B in X, $A \cap B$ is defined as follows:

$$\mu_{A \cap B}(x) = \mu_A(x) \wedge \mu_B(x), \forall_x \in X$$

where "$\wedge$" is the minimum and it corresponds to the connective "and".

The union of the two fuzzy sets, A, B in X, A + B is defined as follows:

$$\mu_{A \vee B}(x) = \mu_A(x) \vee \mu_B(x), \forall_x \in X$$

where "$\vee$" is the maximum and it corresponds to the connective "or".

Some researchers have explored the use of other interpretations of the AND, OR and NOT operations [28]. Here is the example of fuzzy set, TALL and OLD defined by the membership function:

tall(x) = { 0, if height(x) < 5 ft.,
(height(x)-5ft.)/2ft., if 5 ft. <= height (x) <= 7 ft.,
1, if height(x) > 7 ft. }

old (x) = { 0, if age(x) < 18 yr.

(age(x)-18 yr.)/42 yr., if 18 yr. <= age(x) <= 60 yr.

1, if age(x) > 60 yr. }

And for compactness, let

a = X is TALL and X is OLD

b = X is TALL or X is OLD

c = not X is TALL

Then we can compute the following values.

height	age	X is TALL	X is OLD	a	b	c
3' 2"	65?	0.00	1.00	0.00	1.00	1.00
5' 5"	30	0.21	0.29	0.21	0.29	0.79
5' 9"	27	0.38	0.21	0.21	0.38	0.62
5' 10"	32	0.42	0.33	0.33	0.42	0.58
6' 1"	31	0.54	0.31	0.31	0.54	0.46
7' 2"	45?	1.00	0.64	0.64	1.00	0.00
3' 4"	4	0.00	0.00	0.00	0.00	1.00

4.3 Linguistic Variables

Fuzzy logic is primarily concerned with quantifying and reasoning about vague or fuzzy terms that appears in our natural language. In fuzzy logic, these fuzzy terms are referred to as linguistic variables [24].

For example, in the statement "Speed of the computer is fast," we are saying that the implied linguistic variable "**speed**" has the linguistic value of "**fast**". We might assign several linguistic

values for speed such as very slow, slow, medium, fast and very fast. These linguistic values are interpreted as specific fuzzy numbers in which the values are real numbers within a specific range. This is shown in figure 4.2[24].

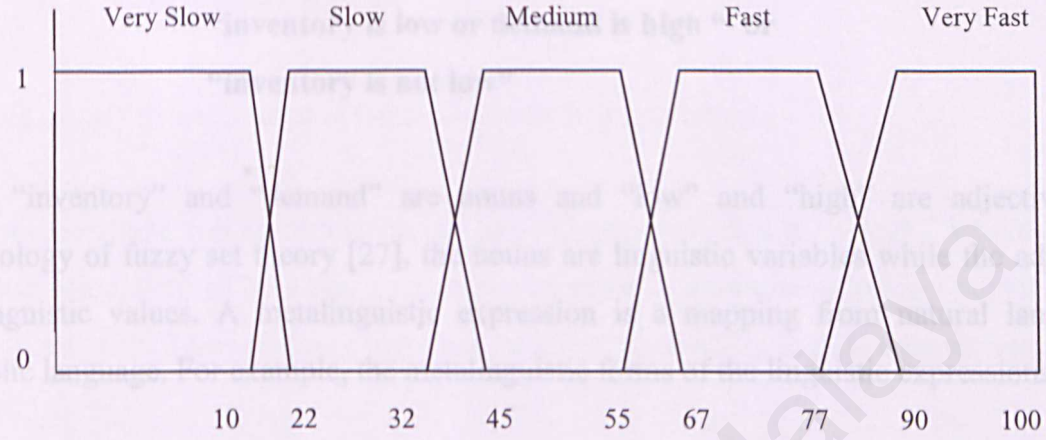

Figure 4.2 Linguistic Values For Linguistic Variable Speed

The examples of linguistic values above show how speed of computer is expressed.

Linguistic variables in fuzzy rules or fuzzy propositions are used in the fuzzy system. Further example of this is shown below [24].

IF speed is slow

THEN the acceleration is high

The range of possible values of a linguistic variable is called the variable's universe of discourse. For example, we might give the value of the variable speed in the range of 0 to 100 mph. The phrase speed is slow occupies a section of the variable's universe of discourse – fuzzy sets.

4.3.1 Linguistic Expressions

Linguistic expressions are the expressions using natural language, for example [27]:

"inventory is low and demand is high,"
"inventory is low or demand is high " or
"inventory is not low"

where "inventory" and "demand" are nouns and "low" and "high" are adjectives. In the terminology of fuzzy set theory [27], the nouns are linguistic variables while the adjectives are the linguistic values. A metalinguistic expression is a mapping from natural language to a symbolic language. For example, the metalinguistic forms of the linguistic expressions above are:

X_1 is A AND X_2 is B,
X_1 is A OR X_2 is B,
X_1 is NOT A,

where X_1 and X_2 are the metalinguistic representations of the linguistic variables, A and B are the metalinguistic representations of the linguistic values, and AND, OR and NOT are the metalinguistic representations of the linguistic connectives [27]. In short form, these metalinguistics are represented by "A AND B," "A OR B," and "NOT A".

4.4 Propositions Of Fuzzy Logic

Fuzzy propositions are created using individual fuzzy sets or groups of fuzzy sets [10]. The power of fuzzy logic comes from its ability to represent vague and subjective concepts. These concepts are generally expressed as linguistic variables. Instead of developing propositions, which depend on the mathematical variables, fuzzy propositions use linguistic variables to express the relationship between the concepts [10]. Fuzzy propositions allow systems to be

created that reason in more human way and easier to understand and maintain. For example, a rule in effort prediction might read:

IF project_size is small
THEN effort is small

This rule is easy to understand in the context of effort prediction.

Fuzzy propositions represent statements like "project complexity is high" where "high" is a linguistic value, defined by a fuzzy set on the universe of discourse of project complexity. Fuzzy propositions is the basis of fuzzy logic and can be combined by logical connectives like *and, or* and *not*.

In fuzzy logic, the truth of propositions is expressed by a fuzzy set through its membership function. A proposition is true or false in two-valued logic while in many-valued logic and fuzzy logic, a proposition p is true to a degree. Linguistic variables in propositions may be modified by linguistic modifiers such as usually, most and very. This is denoted by Q. Propositions may be quantified by the fuzzy concept truth and a fuzzy truth is denoted by T [25].

In the following are the important propositions involving the fuzzy sets [25]

$$\mathbf{A = \{ (x, \mu_A(x)) \} \text{ and } B = \{ (y, \mu_B(y)) \}}$$

1. x = A

   Proposition in canonical form. For example, Proton Wira is a fast car.

2. x = mA

   Modified proposition by the fuzzy modifier. For example, Proton Wira is a very fast car.

3. Qx's are A's

Quantified proposition by the fuzzy quantifier most. For example, Most cars are fast.

4. x is A is T

Qualified proposition by the fuzzy set true. For example, Proton Wira is a fast car is true.

5. If x is A then y is B

Conditional proposition by if-then. For example, If a car is fast then the acceleration is high.

4.5 Fuzzy Rules

A fuzzy rule associates a condition about linguistic variables to a conclusion. Reasoning using fuzzy rule enables an inference to be made from a fuzzy rule even when the rule's condition is only partially satisfied. The degree of the input data matches the condition inferred by the fuzzy rule. The higher the matching degree is, the closer is the inferred conclusion to the rule's consequent.

Fuzzy rules have to be represented by an implication function in order to reason with fuzzy logic. Fuzzy implication has the same function as the classical implication in the classical logic. This implication in classical logic is denoted by [25]

$$A \rightarrow B$$

which can be represented in the statement as

if A then B

In fuzzy logic, this type of statement is referred to as fuzzy if-then statement of fuzzy rules. The **If** part is the antecedent and the **then** is the consequence. The antecedent and the consequence of the fuzzy if-then statement consist of fuzzy proposition as described in section 4.4. The antecedent can contain a combination of propositions by means of the logical connectives "and" and "or". It is also possible that a fuzzy proposition is based on a negation.

A fuzzy rule relates two fuzzy propositions using logical connectives. The example of this is shown below:

if project_size is small
and project_complexity is low
then effort is low

4.6 Fuzzification

Fuzzification is a process of decomposing a system input and/or output into one or more fuzzy sets. Many types of curves can be used to represent fuzzy sets but triangular or trapezoidal shaped membership functions are the most common. The fundamental idea of the process of fuzzification is it allows the inputs and output to be expressed in linguistic term. Figure 4.3 shows a fuzzy set for input with triangular membership functions. A fuzzy set of height has basically three linguistic values: short, medium and tall.

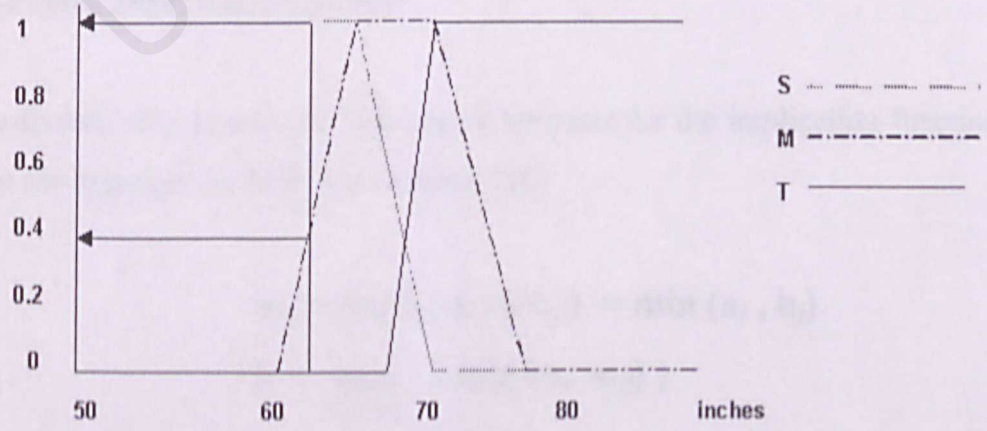

Figure 4.3 Membership Functions for Height Fuzzy Set

A person about 60 inches tall or less is definitely short and the degree membership in short is 1 and the degree of membership in medium and tall is 0. A person about 62 inches tall may be medium with degree of membership of 0.38 but could also be short in degree of membership of 1.

4.7 Fuzzy Inference Method

Inferencing is a process where the relationship between the fuzzy proposition to specific values of the input variables is applied to compute the values of the output variables. Consider a fuzzy rule as follows: [26]

IF X is A THEN Y is B

This rule establishes a relationship between the two propositions. Fuzzy inference attempts to establish a belief in a rule's conclusion given that available evidence on the rule's premise. The two most popular fuzzy inference techniques used in practice are max-min inference and max-product inference methods. These two fuzzy inference techniques will be discussed further in the next section.

4.7.1 Max-Min Inference Method

Max-min inference uses min as the implication operator for the implication function and a max operator for the aggregation. That is as follows: [26]

$$m_{ij} = \text{truth}(a_i \rightarrow b_j) = \min(a_i, b_j)$$

$$b_j = \max_{1 < i < n} \{ \min(a_i, m_{ij}) \}$$

Figure 4.4 shows a schematic representation of fuzzy inference using the max-min inference method. In the event, the input of a rule represents a fuzzy reading, consider the rule IF A THEN B, and a fuzzy reading of A designated as A'. We can simply take the intersection of the two as our input, min (a'_i, a_i) to induce the fuzzy set B'.

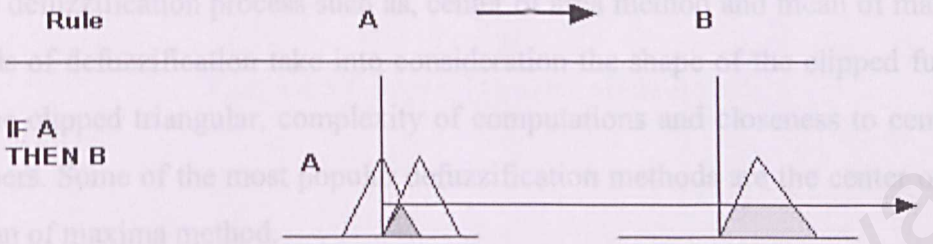

Figure 4.4 Max-min inference [26]

4.7.2 Max-Product Inference Method

The max-product inference method is another commonly applied inference method. Max-product inference uses the standard product as the implication operator when forming the components of M [26]:

$$\mathbf{m_{ij} = a_i\, b_j}$$

Figure 4.5 illustrated the max-product inference technique. Max-min inference produces a clipped version of B while the max-product inference technique produces a scaled version of B.

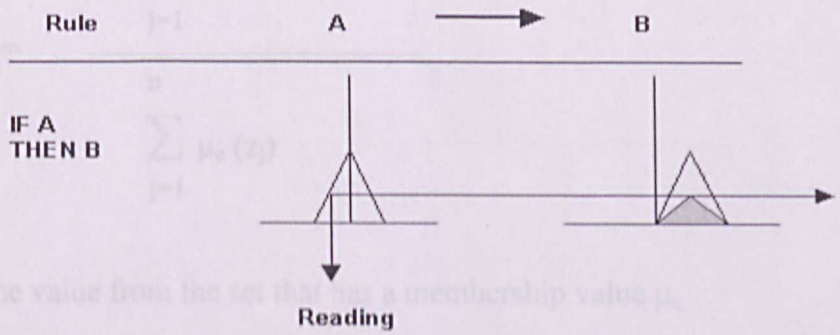

Figure 4.5 Max-product inference [26]

4.8 Defuzzification

Defuzzification is the process of converting each conclusion obtained by the inference engine, which is expressed in terms of a fuzzy set to a single crisp value. There are several methods to perform the defuzzification process such as, center of area method and mean of maxima method. The methods of defuzzification take into consideration the shape of the clipped fuzzy numbers, height of the clipped triangular, complexity of computations and closeness to central triangular fuzzy numbers. Some of the most popular defuzzification methods are the center of area method and the mean of maxima method.

4.8.1 Center Of Area Method

This method is sometimes known as the center of gravity method or centroid method. The center of gravity method is one of the most widely used techniques since it has some advantages. The defuzzified value tends to move smoothly around the output fuzzy region [26]. In this method, the defuzzification value, z_o is defined as the value within the range of variable V for which the area under the graph of membership function c is divided into equal sub-areas. The defuzzified value is calculated by the formula [26]:

$$z_o = \frac{\sum_{j=1}^{n} z_j \cdot \mu_c(z_j)}{\sum_{j=1}^{n} \mu_c(z_j)}$$

where z_j is the value from the set that has a membership value μ_c

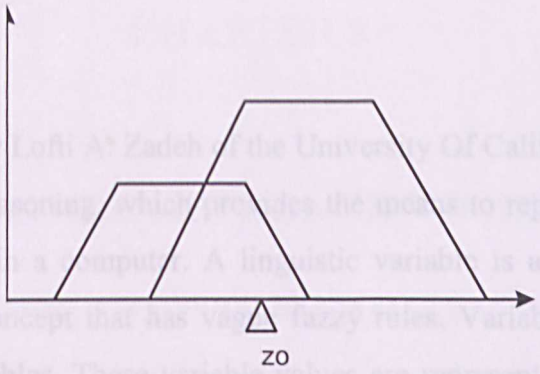

Figure 4.6 Center Of Gravity Method [26]

4.8.2 Mean Of Maxima Method

The defuzzified value z_o is an average of the elements which reach the maximal grade in output fuzzy set C. The defuzzified value is calculated by the formula as follows: [26]

$$z_o = \frac{\sum_{j=1}^{n} z_j}{m}$$

where

z_j is an element giving the maximal grade

m is the number of such maximal elements.

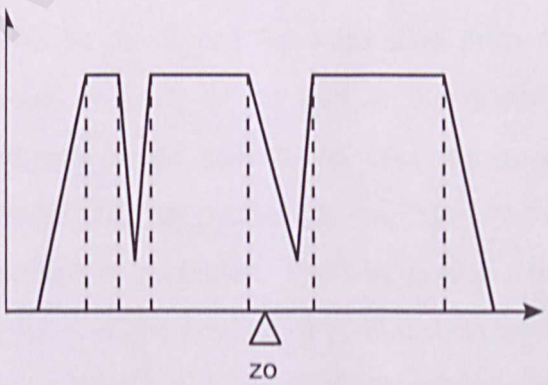

Figure 4.7 Mean of Maxima Method [26]

4.9 Summary

Fuzzy logic was proposed by Lofti A. Zadeh of the University Of California at Berkeley in 1965. It is logic of approximate reasoning, which provides the means to represent and reasoning with vague or ambiguous terms in a computer. A linguistic variable is a term used in our natural language that describes a concept that has vague fuzzy rules. Variables in the fuzzy logic are often referred as fuzzy variables. These variable values are represented using fuzzy sets which map sets of elements to a degree of belief that the elements belongs to the fuzzy set. In classical logic, the truth-values are represented by the crisp set [0,1] (true, false). While in fuzzy logic, the truth-values are represented by membership-values namely, the real numbers in the interval [0,1]. An element has a membership of 1 when it is definitely a member (true) or 0 when it is definitely not a member (false). A membership between 0 and 1 indicates the degree of membership of the element in the set.

Fuzzy logic deals well with uncertain and subjective data. These characteristics suggest that fuzzy logic can be used for effort prediction. The uncertainty comes from the influential factors to cost estimation model such as experience, size of the project, complexity of the project, use of tools, software reliability and total development time. Each factor is therefore represented as a fuzzy linguistic variable.

CHAPTER 5

FUZZY LOGIC APPROACH TO EFFORT PREDICTION MODEL

Several algorithmic models have been proposed to estimate software costs and other management parameters. In chapter 3, the existing traditional cost estimating models such as COCOMO, Function Points and SLIM have been reviewed. As discussed earlier, early prediction of completion time is absolutely essential for proper advanced planning. In this case, fuzzy logic model is used as alternative approach for effort prediction [1,15,29,30,31,32]. It used some form of expert judgement as part of the prediction. The model of the fuzzy logic to effort prediction has been proposed in this chapter. We will look into the input values and the tasks involved in developing the fuzzy model.

5.1 Fuzzy Logic Effort Prediction Model

Generally, cost models have to be developed for estimation purpose. Actual cost estimation depends on the effectiveness and accuracy of the cost model employed. Thus, the parameters used (such as size of the project) in the cost model must be quantifiable or can easily be measured. The problem associated with the parameters employed in the cost model is that, many of them cannot easily be quantified or measured. The best example for this is the experience of the programmer and the complexity of the product. It is almost impossible to measure experience or complexity using percentage or with any mathematical measurement. As such the fuzzy logic model, which is empirically based on reasoning process of a human expert within a specified domain of knowledge, is the best approach for earlier estimation. The use of fuzzy logic models

for software metric modeling seems to be appropriate by using the expert knowledge available from developers and project managers [29,31,32]. Fuzzy logic incorporates a simple rule-based IF...AND...THEN approach in solving problem rather than attempting to model a system mathematically [29].

A fuzzy logic model consists of three components [31]. The first component is the fuzzification process where the membership functions defined on the input variables are applied to their actual values, to determine the degree of truth for each rule premise. The second component is the **if-then** rule base, which can be obtained from expert's understandings of the relationships being modeled. The input membership degree will be mapped between the output membership function. The greater the input membership the stronger the rule fires the output membership functions. There are several consequent of rules fires and different output membership could be contained. Therefore, a defuzzification is carried out. The defuzzification is the third component that combines the outputs into a single label or numerical value as required. Figure 5.1 [31] shows the architectural of fuzzy system.

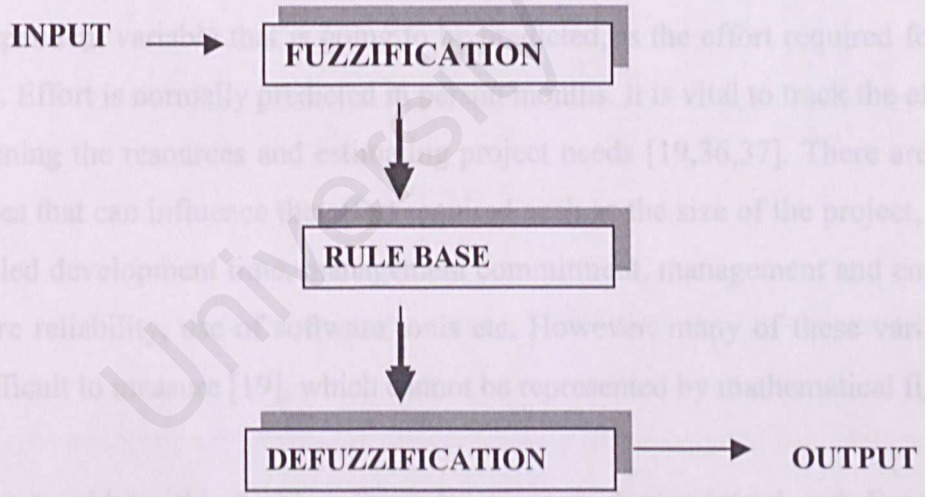

Figure 5.1 An Architectural Of Fuzzy System

Given a set of crisp inputs that represent the fuzzification process converts them into appropriate fuzzy sets and determine their degree of membership in those sets. Then, the rule base will use the fuzzy inputs to determine which rules are applicable to be fired. The rules are independent

and therefore may be evaluated in parallel [31]. The outputs are a set of fuzzy sets defined on the universe of possible outputs. These fuzzy outputs are defuzzified to generate the crisp outputs.

Fuzzy logic model uses a traditional logic theory in modeling a real-world system. A statement such as:

IF (project size is small)
THEN (effort required is small)

would need quantitative definitions for terms. This kind of statement cannot be interpreted in traditional logic. However, fuzzy logic which is based on reasoning process of a human expert, has no problem in interpreting such a statement.

5.2 Variables

The dependent variable that is going to be predicted, is the effort required for development of a project. Effort is normally predicted in person months. It is vital to track the effort for the purpose of planning the resources and estimating project needs [19,36,37]. There are many independent variables that can influence the effort required such as the size of the project, project complexity, scheduled development time, management commitment, management and employee capabilities, software reliability, use of software tools etc. However, many of these variables are subjective and difficult to measure [19], which cannot be represented by mathematical figure.

In order to address this problem fuzzy logic approach was introduced. Fuzzy logic approach is seen and understood as a better alternative as project managers can fairly specify independent variables in software metrics models using linguistic labels in the early stages of estimation [32,39]. Recognizing this approach, our fuzzy logic model uses three independent variables for the prediction of the effort:

1. The size of the project
2. The complexity of the project
3. The scheduled development time

The first factor affecting predicting effort is the size of the project. The measurement of the size is important because the amount of effort required to perform most tasks is directly related to the size of the program involved [37]. This effect seems reasonable when the project size increases, more effort will be required to develop the software product. This effect is shown in figure 5.2[37].

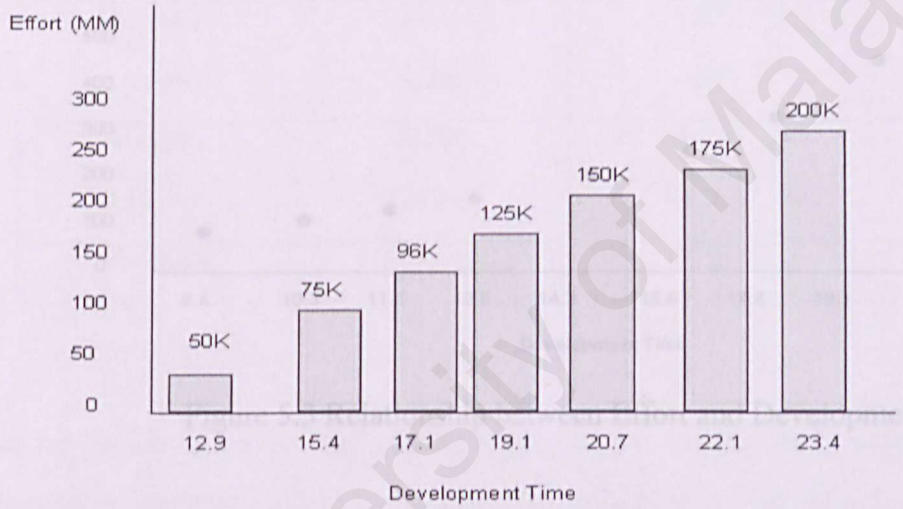

Figure 5.2 Relationship between Effort, Size and Development Time.

The second independent variable is the complexity of the project. It is obvious that some projects are more complex. For example, development for communications software will likely to have a greater complexity than software development for payroll processing. This independent variable also gives data on the complexity of the software code. As the complexity of the software increases, the required development effort increases and the probability of defects increases [37].

Lastly, the development time is a factor that measures the project constraints. If the development schedule of the software project is highly constrained, then the development effort will tends to be high. The purpose of development time measurement is to track the performance of the project team toward meeting the committed schedule. The data from this measure helps management to predict if the original planned delivery dates can be met [37]. Figure 5.3 [37] shows the relationship between development time and effort.

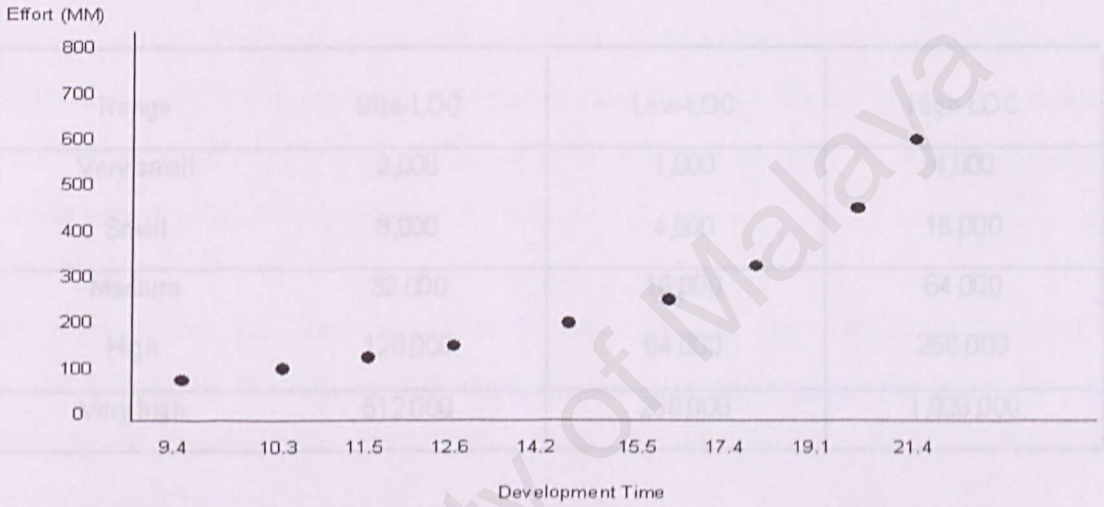

Figure 5.3 Relationship between Effort and Development Time

The purpose of having independent variables to fuzzy logic system is to handle fuzzy linguistic variables for each independent variable. The values of the linguistic variables are represented in the fuzzy sets. A fuzzy set is a function that has several values in a given interval like the unit interval [13]. Suppose for example the complexity of the project can be identified as one of the following linguistic values: *low*, *medium* and *high*. Each linguistic value is assigned to a degree of membership in a fuzzy set. The vagueness and imprecision of the fuzzy linguistic variables are characterized by numerical values using the respective fuzzy sets. The numerical values represent the input values for the fuzzy logic effort prediction.

The fuzzy model for effort prediction is based on fuzzy logic estimating methods. Estimates are often presented as a range. A range is a helpful estimate in planning for predicting effort. For

example in estimating the size of the project, most models require an estimate of size of the project. But this is not measurable in early stage of the development life cycle. Therefore, we need to estimate the size of the project. We could break the size of the project into categories as shown in table 5.1 [36]. To make an estimate, one would judge which of these categories most closely resembles. They should cover the entire span of expected project sizes.

Table 5.1

Fuzzy Logic Size Ranges

Range	Size-LOC	Low-LOC	High-LOC
Very small	2,000	1,000	4,000
Small	8,000	4,000	16,000
Medium	32,000	16,000	64,000
High	128,000	64,000	256,000
Very high	512,000	256,000	1,028,000

The size ranges are subjective and relying mostly on expert judgement based on experience with projects similar characteristics to estimate the size of a project. It is intended for use very early in the software planning process, when the requirements are vague and the design is still undefined. Hence, in order to apply this estimating method, the project manager needs to make two general choices: the overall size of the proposed system and the range within that size [36].

5.3 Building Fuzzy Logic Effort Prediction Model

There are four major tasks that need to perform when developing a fuzzy model [35]:

- Task 1 : Define the linguistic variable
- Task2 : Define the fuzzy sets
- Task3 : Define the fuzzy rules
- Task 4 : Build the system

5.3.1 Defining the Linguistic Variables

In order to illustrate the design of fuzzy logic effort prediction model, the linguistic variables must first be defined. This task is accomplished by uncovering independent and dependent variables. These variables will represent universe of discourse and the fuzzy sets that defined on each of the variables. Generally, linguistic variable is the term used in natural language to describe some concept that usually has vague value.

The fuzzy logic model has three independent variables and one dependent variable. The independent variables are the size, complexity and the scheduled development time of a project. The only dependent variable is the effort. The linguistic variables for fuzzy logic effort prediction model are shown in table 5.2.

Table 5.2

Linguistic Variables for Fuzzy Logic Effort Prediction Model

Linguistic Variable	Range		
Project Size	0K	To	512K
Project Complexity	0	To	10
Development Time	0 Months	To	60 Months
Effort	0 Man-Months	To	1000 Man-Months

5.3.2 Defining the Fuzzy Sets

The next task is to define the fuzzy sets on each linguistic variable. To carry out this task, we need to illustrate a list of typical adjectives used with each linguistic variable. The list is shown in table 5.3. The table represents a vocabulary "dictionary" for the model.

Table 5.3

Fuzzy Variables with Adjectives

Project Size	Project Complexity	Development Time	Effort
Small	Low	Short	Low
Medium	Medium	Medium	Medium
Large	High	Long	High

One possible question that can be raised from the above table is "what is the development time that can be considered as short?" The answer can be around 3 to 10 months. This answer implies the vagueness. This question is used to map the development time values into their corresponding belief values. Fuzzy mapping can have a variety of shapes depending on how the information relates different domain values to belief values. A linear function such as triangular or trapezoidal shape provides an adequate mapping. Figure 5.4 through 5.7 show the fuzzy sets for all terms shown in table 5.3.

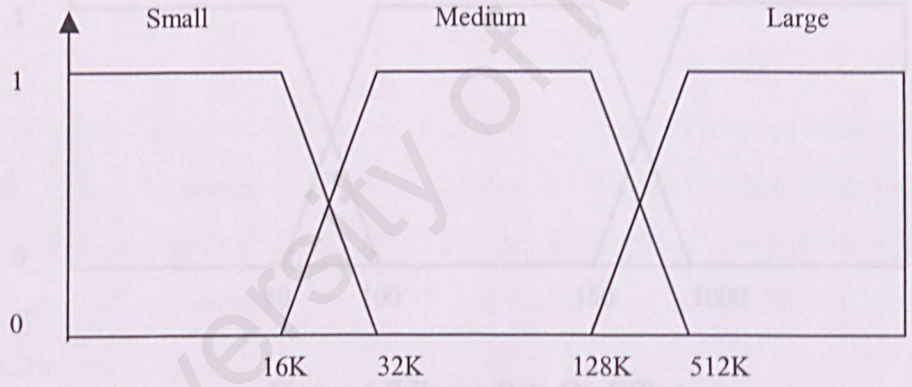

Figure 5.4 Fuzzy Sets On Project Size

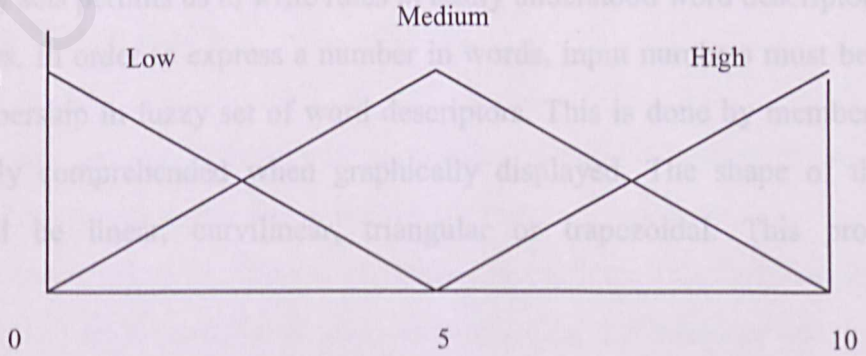

Figure 5.5 Fuzzy Sets On Project Complexity

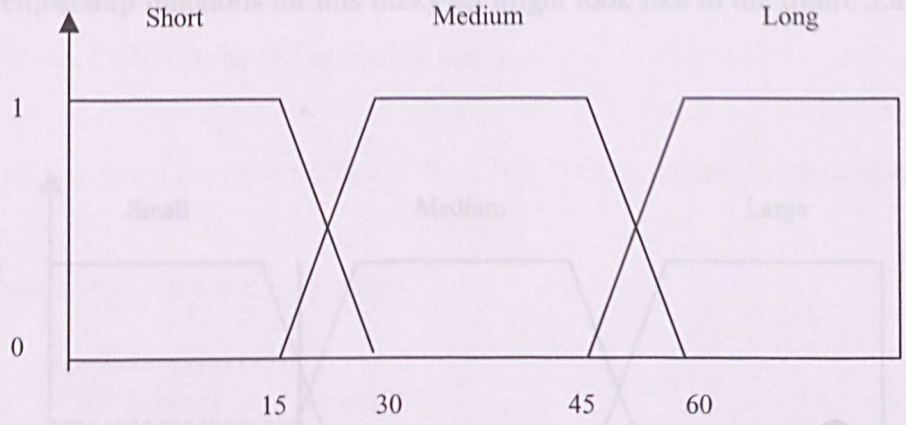

Figure 5.6 Fuzzy Sets On Development Time

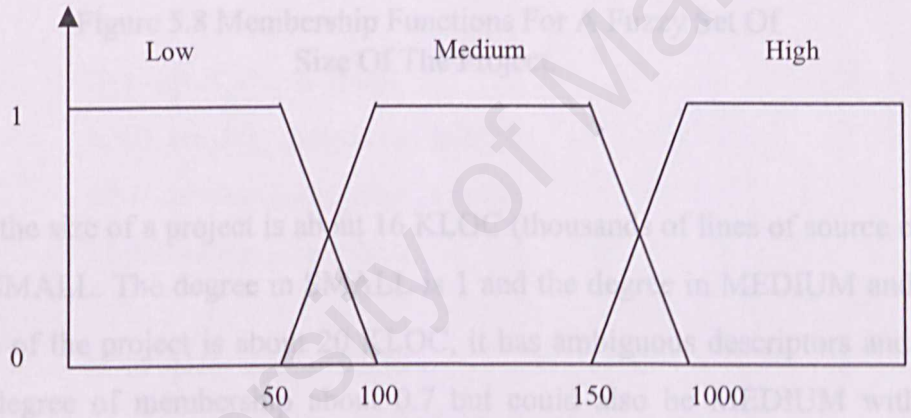

Figure 5.7 Fuzzy Sets On Effort

The use of fuzzy sets permits us to write rules in easily understood word descriptors rather than in numerical values. In order to express a number in words, input numbers must be translated into degree of membership in fuzzy set of word descriptors. This is done by membership functions, which are easily comprehended when graphically displayed. The shape of the membership functions could be linear, curvilinear, triangular or trapezoidal. This process is called fuzzification.

Consider a fuzzy set of the size of the project with three members: SMALL, MEDIUM and LARGE. The membership functions for this fuzzy set might look like in the figure 5.8.

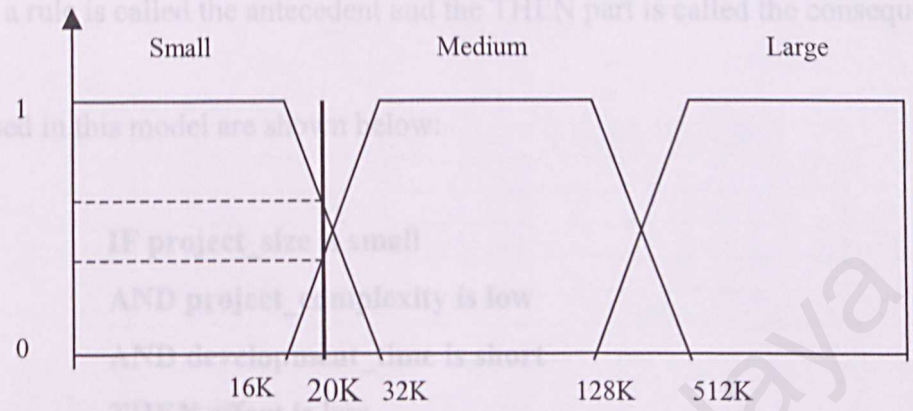

Figure 5.8 Membership Functions For A Fuzzy Set Of Size Of The Project.

In figure 5.8, if the size of a project is about 16 KLOC (thousands of lines of source code) or less it is definitely SMALL. The degree in SMALL is 1 and the degree in MEDIUM and LARGE is zero. If the size of the project is about 20 KLOC, it has ambiguous descriptors and it could be SMALL with degree of membership about 0.7 but could also be MEDIUM with degree of membership of about 0.4.

5.3.3 Defining Fuzzy Rules

In fuzzy logic, rules are also called "production rules" [28,38]. These rules are made up of the antecedent and consequent parts. Some problems produce many rules, only a few of which is actually used in fuzzy inference. This is because only pertinent rules are considered, while the others are ignored. This process, in addition to simplifying the inference process, more closely resembles the way human think. Fuzzy rules are data-driven rules and they are determined by the data. A typical rule, written in English is shown as follows:

IF (the data meet certain specified conditions)

THEN (take the specified actions)

The IF part of a rule is called the antecedent and the THEN part is called the consequent.

Fuzzy rules used in this model are shown below:

IF project_size is small

AND project_complexity is low

AND development_time is short

THEN effort is low.

IF project_size is small

AND project_complexity is high

AND development_time is short

THEN effort is medium.

IF project_size is medium

AND project_complexity is medium

AND development_time is short

THEN effort is high.

The fuzzy input variables in the rules are project_size, project_complexity and development_time. Fuzzy input variables always appear in rule antecedents. A rule consequent refers to one or more fuzzy output variables. The single fuzzy output variable is effort.

The simplest antecedent is of the form "var is adj" where var is a variable name and adj is an adjective that applies to the variable var. In order to evaluate this antecedent, it is necessary to take the value of the membership function at the current value of var. For example, if project_complexity is 5, it is medium to a high degree and if project_complexity is zero or 10 it is

not medium at all. In practice, the fuzzy set have several antecedents that are combined using fuzzy operators such as AND. AND simply uses the minimum weight of all the antecedents. The AND operator is consistent with the Boolean logic as illustrated in the table 5.4 [28].

Table 5.4
Logical AND Operator

A	B	A AND B min(A,B)
0	0	0
0	1	0
1	0	0
1	1	1
0.9	0.1	0.1

5.3.4 Building the System

After defining the fuzzy sets and fuzzy rules, we can now build the system. This task involves coding of the fuzzy sets and rules and procedures for performing fuzzy logic functions. The system is built using the Prolog programming language [40,41,42,43]. Prolog is a logic-based programming language, which consists of a series of rules and facts, expressed in the syntax of Prolog. The performance of the system is addressed in the next chapter.

5.4 Advantages of Fuzzy Logic

Fuzzy logic approach to software metrics effort prediction provides considerable benefits in terms of reducing commitment, the use of fuzzy labels as independent variables, making full use of knowledge and improving interpretability [1,31,32].

5.4.1 Fuzzy Linguistic Variables as Independent Variables

Since the independent variables in the software metric models such as complexity of the project and the size of the project are difficult to quantify and only known to a rough degree, the use of fuzzy linguistic variables seem intuitively appealing. Here, project managers are in fact able to

classify the influential factors of effort prediction with reasonable levels of consistency. For example the linguistic variables for the effort prediction are the size of the project, the complexity of the project, development time and effort. The values of the fuzzy linguistic variables can be large, small, medium, low, high or long. These values will be mapped into fuzzy sets and the defuzzification process will be done to obtain one crisp output value.

5.4.2 Reducing Commitment through Fuzzy Outputs

Estimating effort at the early stage of software development project is not realistic. Instead, a fuzzy system may be used to transform linguistic values or numerical values, indicating project size, project complexity and development time into an equally imprecise (but adequate for its purpose) label indicating predicted effort, for example very high.

5.4.3 Better Use Of Knowledge and Data

Since fuzzy logic can be initialized with expert rules, and given that the movement of membership functions and rules can be limited, it is possible that such a model will perform significantly better than alternatives such as regression models in small quantities of data [8].

5.5 Summary

Software cost estimation deals with vague concepts such as "size of the project" and "complexity of the project" that are difficult to measure numerically. The use of fuzzy logic as alternative for software cost estimation can deal with these vague concepts using fuzzy linguistic variables. In this sense, instead of using numerical input, the project manager is able to define the input variables using linguistic terms such as high complexity or medium size of the project. It is considered that fuzzy system offers several benefits in terms of reducing commitment and making better use of knowledge and data. The motivation for this has been the difficulties faced by software metricians in terms of avoiding premature and costly commitment [1].

CHAPTER 6

SYSTEM IMPLEMENTATION AND PERFORMANCE

The cost influential factors in the existing cost estimation models are found to be imprecise and vague. However, fuzzy logic provides an easy way of dealing with the vagueness and impreciseness of the influential factors. It can be built with rules containing common sense concepts as "high", "medium" and "low" especially for applications where the input values are numerical. This chapter describes the implementation of fuzzy logic engine in predicting the effort, which is written in Prolog. The fuzzy logic engine includes methods to compile fuzzy rules, define the input values, define the fuzzy sets, execute the fuzzy rules and evaluate the value of fuzzy variables. Under this chapter, we will describe in more detail the development of FLEP program using the fuzzy logic approach and Amzi! Prolog tool.

6.1 Tool Selection

FLEP (Fuzzy Logic Effort Prediction) program is written in Prolog language. Prolog (PROgramming in LOGic) was created by Colmerauer and his colleagues at the University of Marseilles in 70s [40,41,42,43]. The difference between Prolog and other languages is that a Prolog program tells the computer what to do (a technique called declarative programming) while program in other languages tell the computer how to do (procedural programming). Generally, prolog is a logic-based computer, which allows the programmer to define rules, facts and makes queries. In English, facts are statement like 'The effort required is high' or 'The project size is small'. Rules are statements like 'If project size is small then effort required is low'. A rule is characterized by 'if' and connecting some conclusion with one or more premise. Prolog deals with logic directly and allows to store and use facts and rules.

Prolog system consists of Prolog interpreters and Prolog compilers. An interpreter stores the program in a form like original text and translates them into instructions and actions, and it can be done repeatedly. A compiler translates all the source code into machine language only once. This allows the code to be executed immediately without the interpretation stage. Interpreter is easy to implement but much slower than compilers because of the repeated translation of the code that is necessary.

Prolog offers several significant advantages in developing the fuzzy logic cost estimation program. First, the power of programming in Prolog is achieved by the use of rules [39,40]. The format of the rules is of course the rule language. Typically, they are expressed in an 'if-then' syntax. The rules refer to the data that are to be activated. There are three key technologies needed for implementing a rule

- Language parsing tools for the rule language
- Pattern-matching tools to determine if a rule can fire
- Search mechanisms for efficiently finding the right rules

While these tools can be developed in any language, they are an integral part of the Prolog language.

Secondly, the data representation in Prolog [39,40]. Rules always express relationships between data elements. But how is the data represented? The simplest approach is by using attribute value pairs as shown in the example below:

IF project_size is large

THEN effort is high

Project_size attribute could have various size values. Attributes and their values can be easily represented using Prolog's built-in database. For example, if the size of the project was determined to be small, this Prolog statement would execute:

Assert (known (project_size, small))

For this reason, Prolog is used to build the fuzzy logic effort estimation program. We used Amzi! Prolog to write the fuzzy logic cost estimation program. Amzi! Prolog is a powerful implementation of the Prolog language, this is because its integrated development environment (IDE) which includes an editor, interpreter (listener) and debugger for developing Prolog modules and a compiler, projects and linker for deploying them. Figure 6.1 shows the Amzi! Prolog Architecture [40].

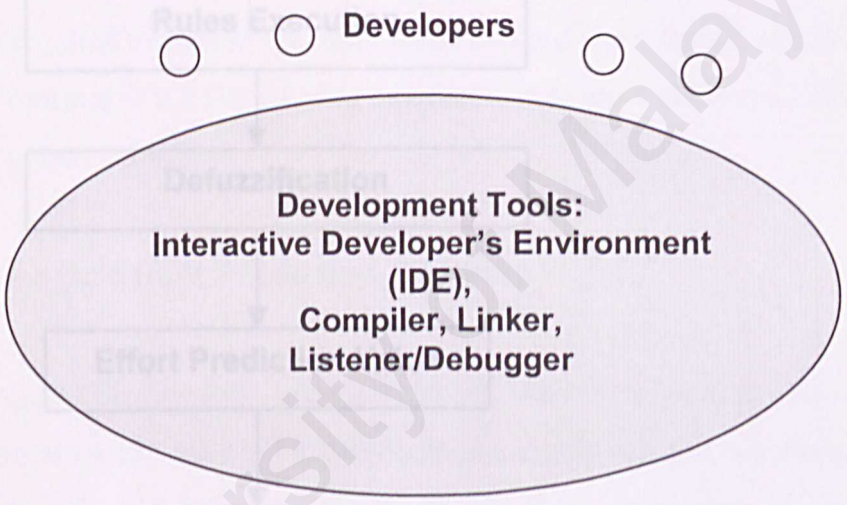

Figure 6.1 Amzi! Prolog Architecture

Amzi! Prolog has a user-friendly IDE with a concise and well-organized documentation. It is embedded in a powerful programming environment [40].

6.2 Fuzzy Logic Approach With Amzi! Prolog

The program flow chart given in figure 6.2 shows how the Amzi! Prolog is being used in the fuzzy logic approach for effort prediction.

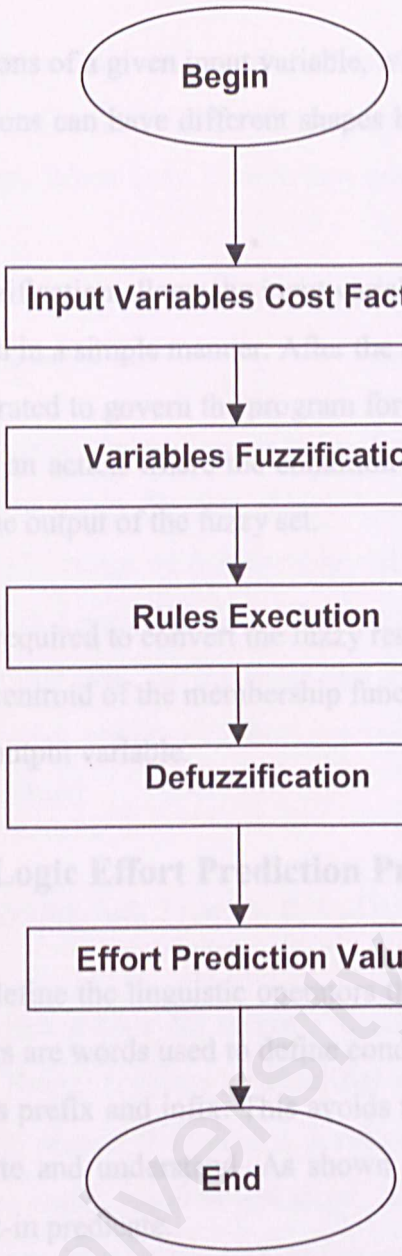

Figure 6.2 Flow Chart Of Fuzzy Logic Cost Estimation Program.

In the input variables cost factors process, the user will enter the value of the size of the project, the project complexity and total development time. The entered input variables will then be fuzzified. Fuzzification process is the process of decomposing the input variables into one or more fuzzy sets. The input variables will be fuzzified into a different degree of membership functions in each fuzzy set. A degree of membership in a set is based on a scale from 0 to 1 with 1 being complete membership and 0 being no membership. An output is calculated based on the

membership functions of a given input variable, which has been configured in the fuzzy sets. The membership functions can have different shapes but the most common forms are triangular and trapezoid.

The process of fuzzification allows the input variables to be expressed in linguistic terms so those rules can be applied in a simple manner. After the inputs have been decomposed into fuzzy sets, a set of rules is generated to govern the program for each combination of inputs. Each rule consists of a condition and an action where the condition is interpreted from the input and the output is determined from the output of the fuzzy set.

Defuzzification is required to convert the fuzzy result into one exact output value which is done by computing the centroid of the membership function of the output fuzzy set. Effort prediction value is the fuzzy output variable.

6.3 The Fuzzy Logic Effort Prediction Program

It is necessary to define the linguistic operators that are used to define fuzzy rule before coding. Linguistic operators are words used to define conditions, conclusion and negations in fuzzy rules, which is defined as prefix and infix. This avoids the use of lots of brackets and makes the fuzzy rules easier to write and understand. As shown below, the linguistic operators are defined by using the 'op' built-in predicate.

```
% Linguistic Operator Definitions

:- op(700, xfx, is).
:- op(720, xfy, and).
:- op(740, xfy, or).
:- op(760, xfx, then).
:- op(780, fx,  if).
```

The 'if' is for constructing rules and it is prefix that comes before the premises and the conclusion. The premises and the conclusion are connected by the infix operator 'then' which comes between them. When there is more than one premise, they are joined by the infix operator 'and'.

6.3.1 Defining Fuzzy Sets

A fuzzy set has a degree of membership for each value in its domain. The degree of membership is given by the membership function, which has a value between 0 and 1. The interpretations, which have value of 0, means no membership and 1 denotes complete membership and a value in between them denotes a partial or uncertained membership. The membership functions can have different shapes. In this case, the trapezoid and triangular forms are used. The fuzzy sets can be the project_size, project_complexity, development_time or effort. Below is the example of how the fuzzy sets are defined:

Fuzzy_set(Name, Set, Type, A, B, C, D)

This function provides a hook for creating a user-defined fuzzy set. The parameters are totally user-defined. The **'Name'** contains an identifier denoting a user-defined name for the fuzzy set such as project_size, project_complexity, development_time and effort. The **'Set'** is the name of the qualifier such as small, medium and large. **'Type'** is the shape that is used to implement the qualifiers. In this case, **tp** (trapezoidal), **at** (ascending trapezoidal/triangular) and **dt** (descending trapezoidal/triangular) forms are used. The geometrical shape of the trapezoid and triangular are given by **A, B, C** and **D**. Table 6.1 shows the values of A, B, C and D.

Table 6.1

Values of A, B, C and D

Trapezoidal		Triangular	
A	Minimum Value	A	Minimum Value
B	Left Shoulder	B	Center
C	Right Shoulder	C	Center
D	Maximum Value	D	Maximum Value

The definition of a fuzzy set may also include a definition of the range of 'crisp' values that it may take. The range of project_size for effort estimation may be defined as being between 0 and 512K (measured in thousand of lines of code). The project_size of the effort estimation is normally described using the adjectives such as small, medium and large. In order to implement any qualifier, we need to define it for the 'project_size' fuzzy set. The definition of the qualifier usually involves trapezoid, triangular and linear shape. The following defines the fuzzy set for project size.

fuzzy_set(project_size, small, tp, 0.0, 0.0, 16.0, 32.0).

fuzzy_set(project_size, medium, tp, 16.0, 32.0, 128.0, 512.0).

fuzzy_set(project_size, large, at, 128.0, 512.0, 0.0, 0.0).

The 'project_complexity' input can also be defined in a similar way. The project_complexity of the effort estimation is normally described using the adjectives: low, medium and high. The range of project_complexity may be defined as being between 0 to 10. The result is the following definition for the 'project_complexity' fuzzy set:

fuzzy_set(project_complexity, low, dt, 0.0, 5.0, 0.0, 0.0).

fuzzy_set(project_complexity, medium, tp, 0.0, 5.0, 5.0,10.0).

fuzzy_set(project_complexity, high, at, 5.0, 10.0, 0.0, 0.0).

Another input for this program is 'development_time', which can be defined in a similar way. The adjectives for the development_time are short, medium and long and the range of it may be defined as being between 0 to 60. The following definitions are the result for the 'development_time' fuzzy set:

fuzzy_set(development_time, short, tp, 0.0, 0.0, 15.0, 30.0).
fuzzy_set(development_time,medium, tp,15.0,30.0,45.0,60.0).
fuzzy_set(development_time, long, at, 45.0, 60.0, 0.0, 0.0).

The final fuzzy set in this program is the 'effort' which is the output of the fuzzy set. We define the range for the 'effort' as being between 0 and 1000 and the adjectives for the effort are low, medium and high. This time we will also need to consider how to return a 'crisp' value from the variable at the end of the fuzzy program. As a reasonable starting point we will use the default 'centroid' method. This results in the following definition for the 'effort' fuzzy set:

fuzzy_set(effort, low, tp,0.0, 0.0, 50.0, 100.0).
fuzzy_set(effort, medium, tp, 50.0, 100.0, 150.0, 1000.0).
fuzzy_set(effort, high, at, 150.0, 1000.0, 0.0, 0.0).

6.3.2 Defining Fuzzy Rules

The rules in this program will cover all the possible combinations of project_size, project_complexity and development_time. The complete set of rules for the problem would then be as follows:

%::: PRODUCTION RULES (FUZZY MODEL'S RULES) :::

% R-1
if project_size is small
and project_complexity is low
and development_time is short
then effort is low.

% R-2
if project_size is small
and project_complexity is medium
and development_time is short
then effort is low.

% R-3
if project_size is small
and project_complexity is high
and development_time is short
then effort is medium.

% R-4
if project_size is small
and project_complexity is low
and development_time is medium
then effort is low.

% R-5

if project_size is small

and project_complexity is medium

and development_time is medium

then effort is medium.

% R-6

if project_size is small

and project_complexity is high

and development_time is medium

then effort is medium.

% R-7

if project_size is small

and project_complexity is low

and development_time is long

then effort is low.

% R-8

if project_size is small

and project_complexity is medium

and development_time is long

then effort is medium.

% R-9

if project_size is small

and project_complexity is high

and development_time is long

then effort is medium.

% R-10

if project_size is medium

and project_complexity is low

and development_time is short

then effort is medium.

% R-11

if project_size is medium

and project_complexity is medium

and development_time is short

then effort is high.

% R-12

if project_size is medium

and project_complexity is high

and development_time is short

then effort is high.

% R-13

if project_size is medium

and project_complexity is low

and development_time is medium

then effort is medium.

% R-14

if project_size is medium

and project_complexity is medium

and development_time is medium

then effort is medium.

% R-15
if project_size is medium
and project_complexity is high
and development_time is medium
then effort is high.

% R-16
if project_size is medium
and project_complexity is low
and development_time is long
then effort is medium.

% R-17
if project_size is medium
and project_complexity is medium
and development_time is long
then effort is medium.

% R-18
if project_size is medium
and project_complexity is high
and development_time is long
then effort is high.

% R-19
if project_size is large
and project_complexity is low
and development_time is short
then effort is high.

% R-20
if project_size is large
and project_complexity is medium
and development_time is short
then effort is high.

% R-21
if project_size is large
and project_complexity is high
and development_time is short
then effort is high.

% R-22
if project_size is large
and project_complexity is low
and development_time is medium
then effort is high.

% R-23
if project_size is large
and project_complexity is medium
and development_time is medium
then effort is high.

% R-24
if project_size is large
and project_complexity is high
and development_time is medium
then effort is high.

```
% R-25
if project_size is large
and project_complexity is low
and development_time is long
then effort is high.

% R-26
if project_size is large
and project_complexity is medium
and development_time is long
then effort is high.

% R-27
if project_size is large
and project_complexity is high
and development_time is long
then effort is high.
```

The rules depend on data acquired from the user. The Prolog code below is used to ask data from the user. It uses two Prolog rules. The first rule checks to see if an attribute already has a known value, and if so compares it to the required value and the result is either succeed or fail. If there is no previous known value, then the second rule fires, asking the user, saving the answer and again checking to see if the answer was as expected.

```
ask(Attr, Value) :-
 known(Attr, X),   % is there a known value for this attribute?
 !,                % if so don't ask again
 X = Value.        % succeed or fail based on the expected value.

ask(Attr, Value) :-
 write('What is the value of '),
```

```
write(Attr), write('? '),      % ask the user
read(X),                       % get the answer
assert( known( Attr, X ) ),    % remember it
X = Value.                     % succeed or fail based on the value
```

As the rule engine starts to look for rules it will need to ask the user about the size of the project, the complexity of the project and the total development time. It will only ask once and when testing, other rules will use the remembered answer to the question.

This type of user-interaction is particularly important for prediction system, to determine the cause of a problem based on a dialog with the user. By using a predicate such as 'ask' a system can be designed to only ask questions that are triggered by pertinent rules and only ask when the data is actually needed.

6.3.3 The Remaining Fuzzy Programs

The remaining fuzzy programs will wrap up the fuzzy sets and rules in a program that can be run. This program will set the inputs for the 'project_size', 'project_complexity' and 'development_time' and decide how to propagate the degrees of membership using the fuzzy rule. The result for the effort fuzzy set will only be produced after the propagation.

Init(effort) will initialize the fuzzy logic engine and **one_goal(effort)** shows that this program will give one goal (effort required) at a time. These two functions are the main procedure of the program.

In the function fuzzification, the fuzzy input values are translated into degree of membership in a fuzzy set. The fuzzification function is shown as follows:

```
%::: FUZZIFICATION :::
fuzzification( N, Cj, X ) :-
        fuzzy_set( N, Cj, T, A, B, C, D ),
        degree_of_membership( T, A, B, C, D, X, M ),
        assert(prem(M)), !.
```

The membership function is evaluated in the degree_of_membership function. A fuzzy variable gets its membership function from one or several fuzzy sets. In other words, the membership functions from the fuzzy sets are combined into one membership function.

```
% LINEAR DECREASING FUZZY SET (dt)
degree_of_membership( dt, A, _, _, _, X, 1.0 ) :-
        X =< A, !.
degree_of_membership( dt, _, B, _, _, X, 0.0 ) :-
        X >= B, !.
degree_of_membership( dt, A, B, _, _, X, M ) :-
        line_eq( dt, A, B, X, M ), !.

% LINEAR INCREASING FUZZY SET (at)
degree_of_membership( at, A, _, _, _, X, 0.0 ) :-
        X =< A, !.
degree_of_membership( at, _, B, _, _, X, 1.0 ) :-
        X >= B, !.
degree_of_membership( at, A, B, _, _, X, M ) :-
        line_eq( at, A, B, X, M ), !.

% TRAPEZOIDAL OR TRIANGULAR FUZZY SET
degree_of_membership( tp, A, _, _, _, X, 0.0 ) :-
        X =< A, !.
degree_of_membership( tp, A, B, _, _, X, M ) :-
```

```
        X > A, X =< B,
        line_eq( at, A, B, X, M ), !.
degree_of_membership( tp, _, B, C, _, X, 1.0 ) :-
        X > B, X < C, !.
degree_of_membership( tp, _, _, C, D, X, M ) :-
        X > C, X < D,
        line_eq( dt, C, D, X, M ), !.
degree_of_membership( tp, _, _, _, _, _, 0.0 ).  % X>D
```

Function fuzzy operators define the fuzzy operators used in this program. The function of fuzzy operators is shown below

```
%::: FUZZY OPERATORS :::
apply_fuzzy_oper( and_z ) :-
        retract(prem(M1)),
        write(M1),
        retract(prem(M2)), !,
        write(' and '), write(M2), nl,
        min(M1,M2,M),
        assert(prem(M)), !.

apply_fuzzy_oper( or_z ) :-
        retract(prem(M1)),
        write(M1),
        retract(prem(M2)),!,
        write(' or '), write(M2), nl,
        max(M1,M2,M),
        assert(prem(M)), !.
```

Function centroid defuzzification method defines the process of the defuzzification. Defuzzification is a transformation of the fuzzy variables to a real value by taking the centroid of

the fuzzy variable. An equality comparison between a real value x/s and a fuzzy variable is done by evaluating the membership function at x/s. The result is in the range of 0 to 1 and denotes the membership of x/s in the fuzzy set. The fuzzy logic engine handles both Boolean logic and fuzzy logic by taking the minimum value for an AND and the maximum value for an OR.

```
%::: CENTROID DEFUZZIFICATION METHOD :::
centroid(Var,Outset,Memb) :-
        fuzzy_set(Var,Outset,_,S,_,_,_),!,
        P is (S*Memb),
        retract(sum1(Var,Q)),
        R is (P+Q),
        assert(sum1(Var,R)),
        retract(sum2(Var,N)),
        M is (Memb+N),
        assert(sum2(Var,M)),
        write(Var),write(' = '),write(S),
        write('  membership = '),write(Memb),nl,
        write('Centroid: '),write(Var),write(' = '),write(R/M),nl,!.
```

The fuzzy logic engine has a rich syntax that is able to handle the prolog interpreter including some common predefined functions. The functions below show the prolog interpreter and other predefined functions in this program.

```
prove(ATTR is VALUE and REST) :-    % AND
getav(ATTR, VALUE),
prove(REST),
apply_fuzzy_oper(and_z).
```

```
prove(ATTR is VALUE or REST) :-          % OR
getav(ATTR, VALUE),
prove(REST),
apply_fuzzy_oper(or_z).

prove(ATTR is VALUE) :-                  % IS
getav(ATTR,VALUE).

getav(ATTR,VALUE) :-                     %IF/THEN (CONCLUSION)
if CONDITIONS then ATTR is VALUE,
prove(CONDITIONS),
retract(prem(Mx)),
centroid(ATTR,VALUE,Mx).

getav(ATTR,VALUE) :-
not(if _ then ATTR is _),
rule_translation(ATTR,VALUE).
```

The rules of 'prove' shown in the prolog interpreter cover three cases.

(1) If there is a list of sub-goals to prove, separated by 'and's, then call getav to see if the first attribute value pair is true, and, if so, prove the rest.

(2) If there is a single negated goal, simply ask for the value and negate it.

(3) If there is a single goal, call getav to see if its true. The rules of 'getav' cover two cases.

(i) If the attribute is defined in the 'then' side of a rule, try to prove the 'if' conditions of the rule.

(ii) If the attribute is not on the 'then' side of any rule, the user will be asked.

6.4 Running the Fuzzy Logic Effort Prediction Program (FLEP)

To run the Fuzzy Logic Effort Prediction (FLEP) program, simply consult into a Prolog interpreter 'Listener' and choose 'start' from the pull-down menu as shown in the figure 6.3.

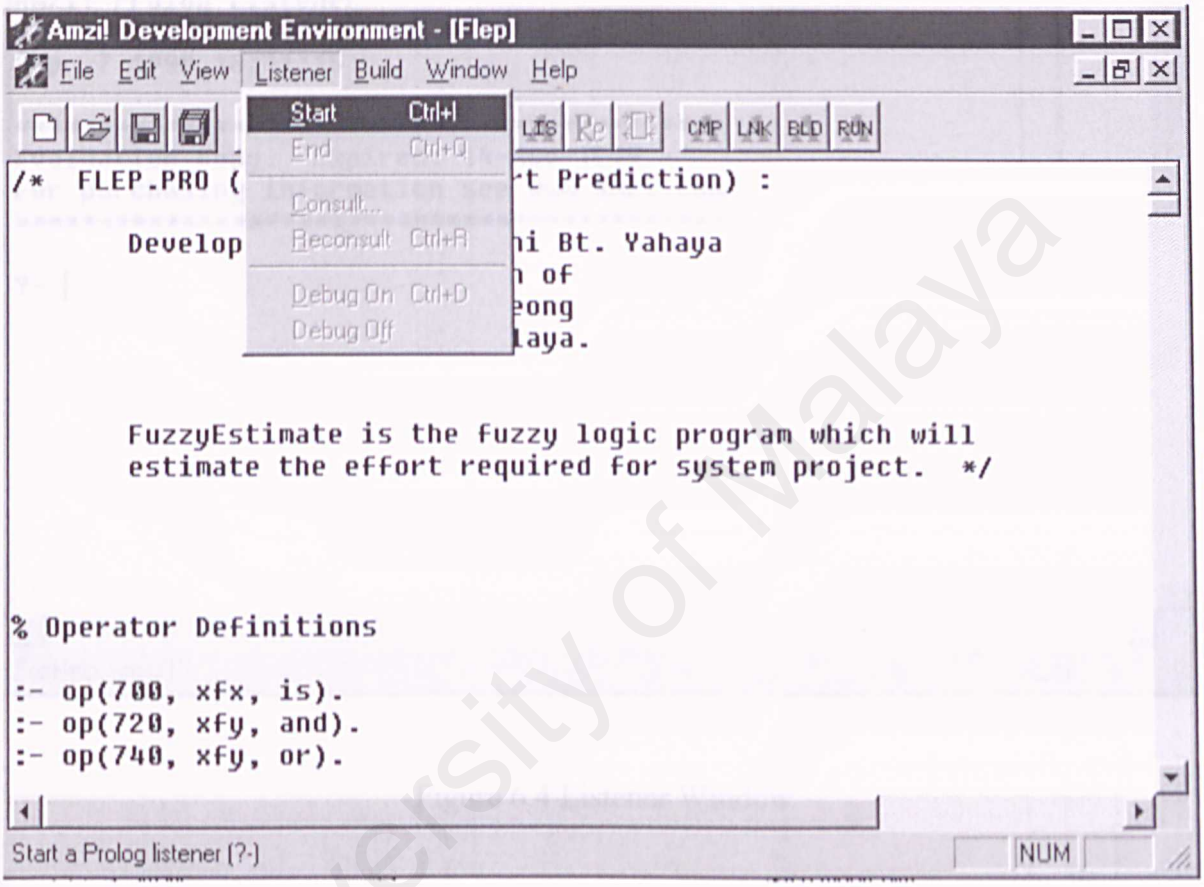

Figure 6.3 Listener Pull-down menu

After choosing the start option, listener window will appear as shown in the figure 6.4.

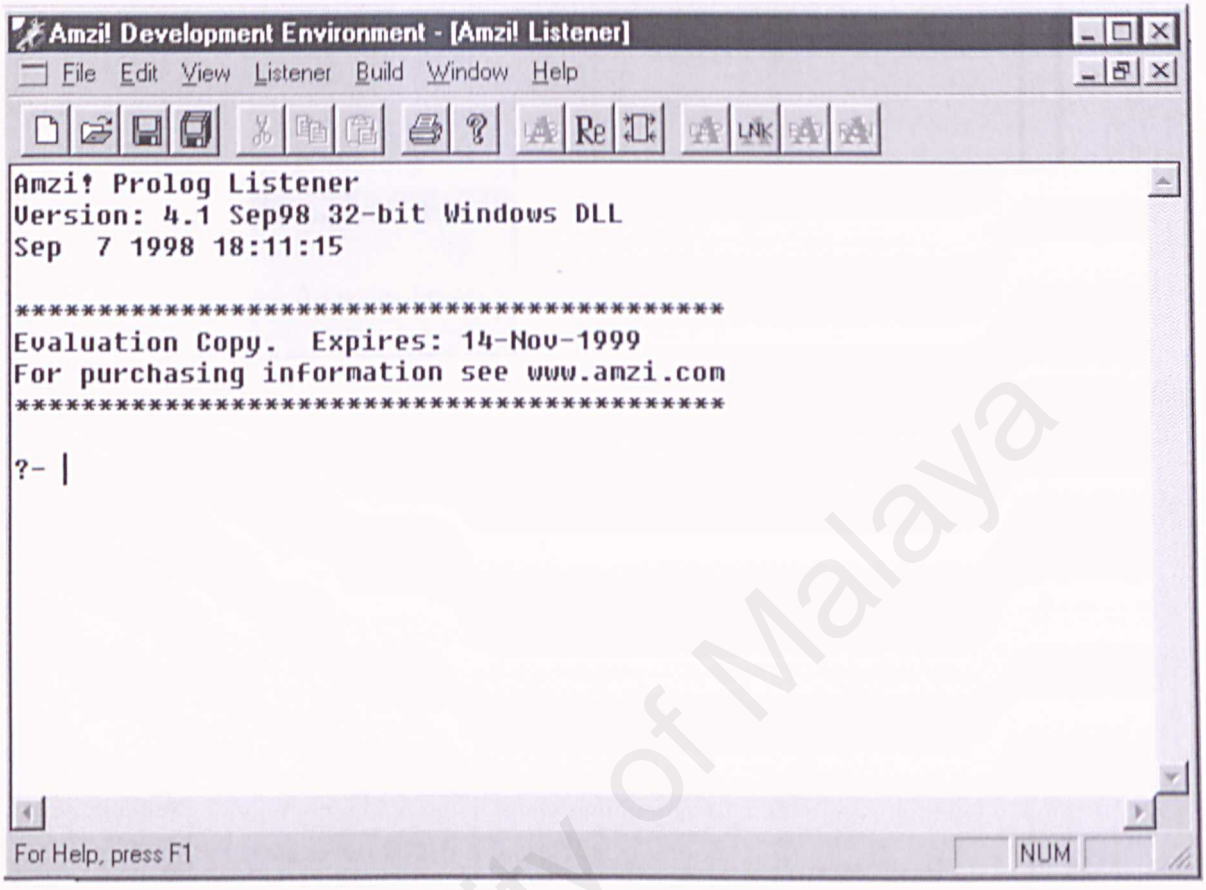

Figure 6.4 Listener Window

The FLEP program is stored in a Prolog database and it is loaded using the consult command. Figure 6.5 shows the consult command.

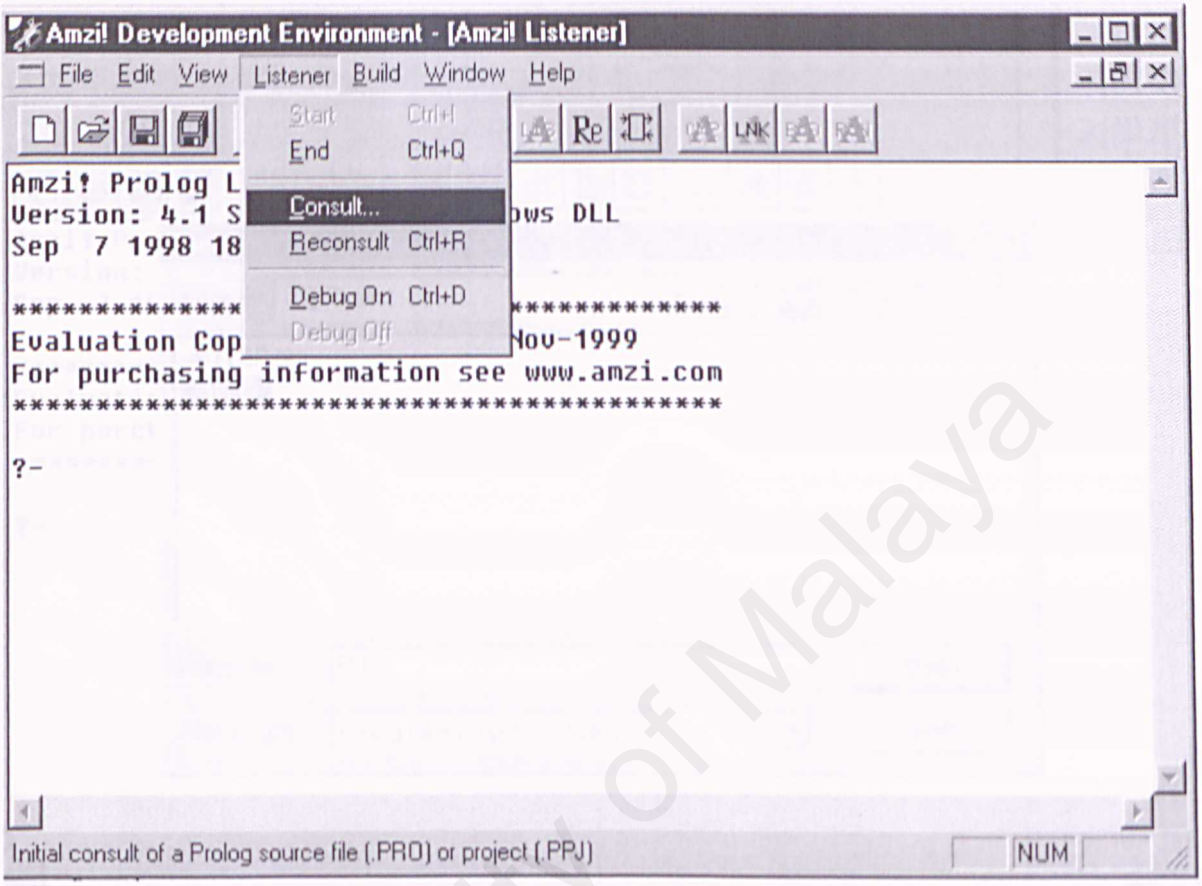

Figure 6.5 Consult Command

The name of the file must be specified in order to load the program as shown in the figure 6.6.

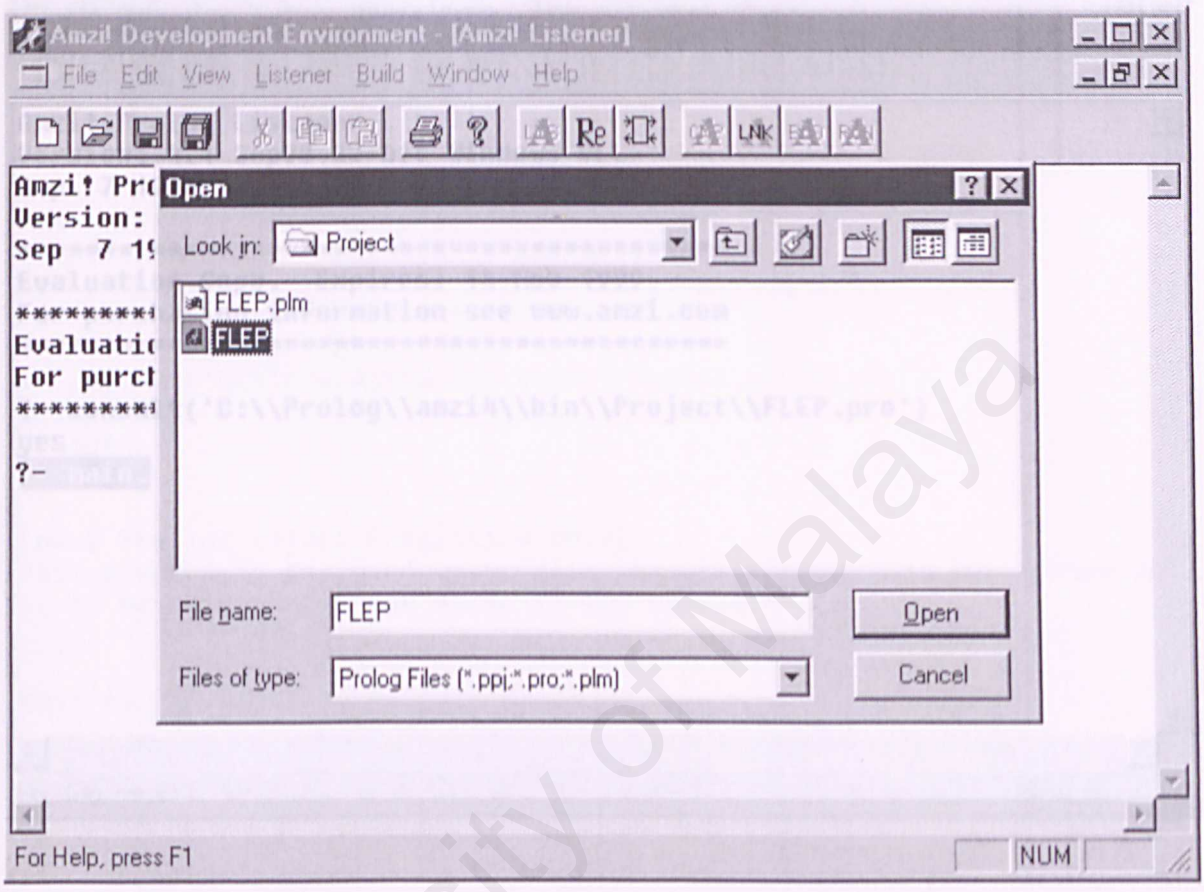

Figure 6.6 Open File

The FLEP program is started by typing '**main.**' shown in the figure 6.7.

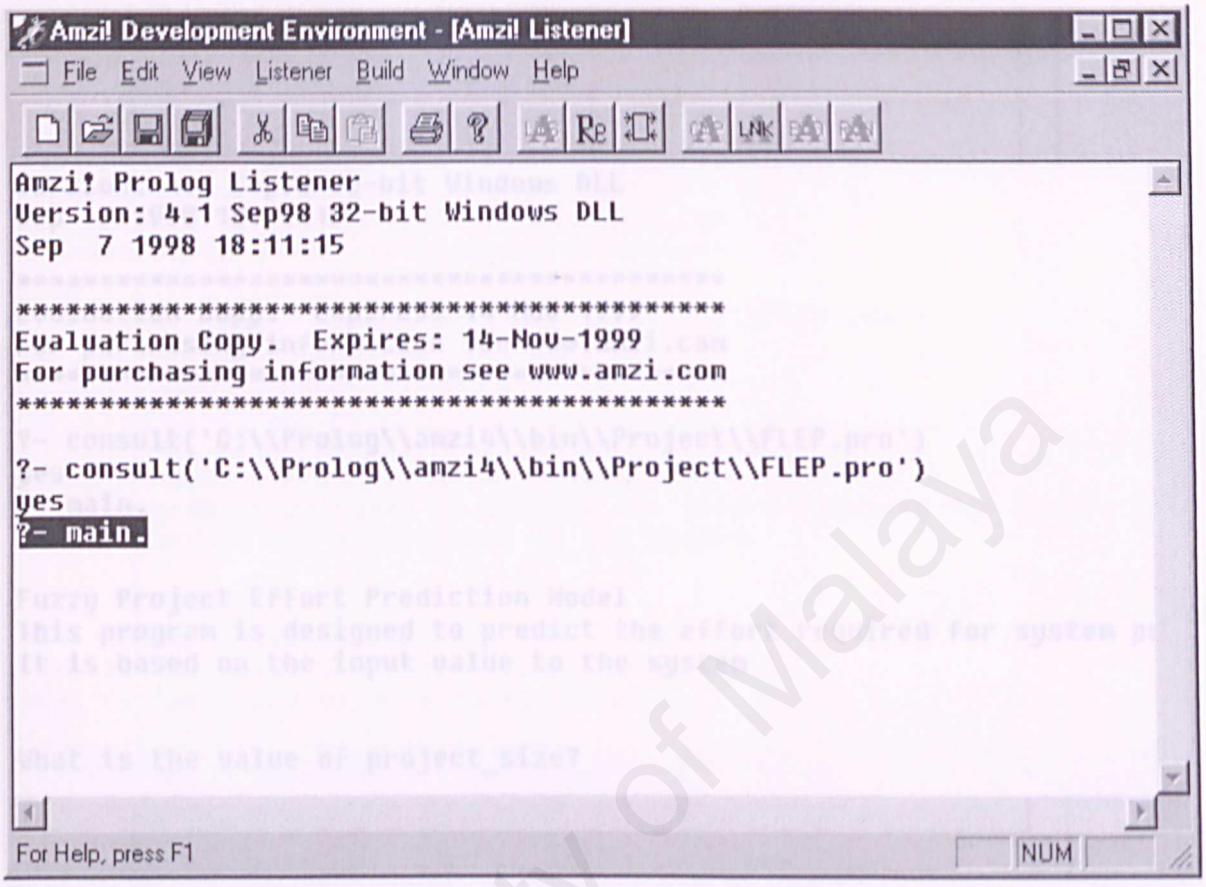

Figure 6.7 Start FLEP program

The output of the program is shown in the figure 6.8.

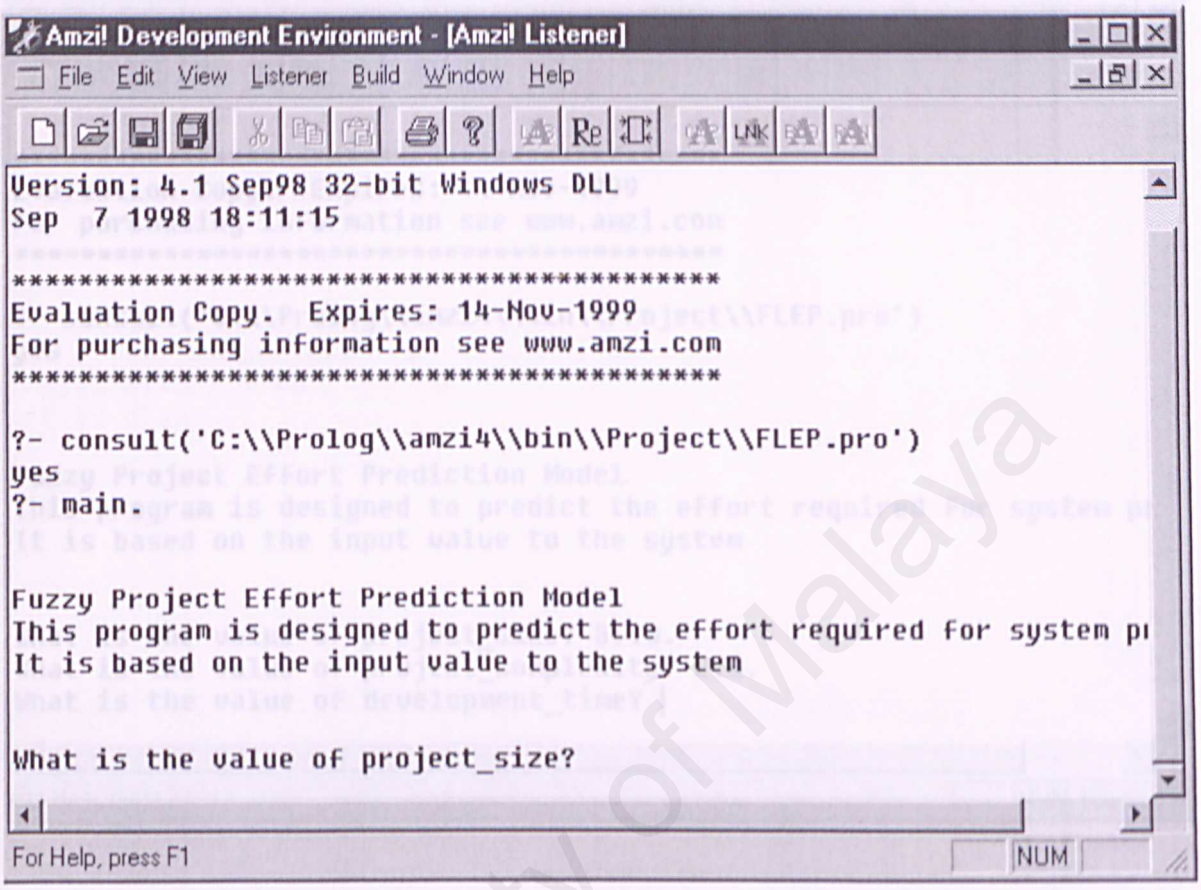

Figure 6.8 Output of the Program

The user must insert a value for the project size ranging from 0 to 512 thousands of lines of code. After the user has inserted the value for the project size and presses enter, the user must insert input values for both the project complexity and development time. This is shown in the figure 6.9.

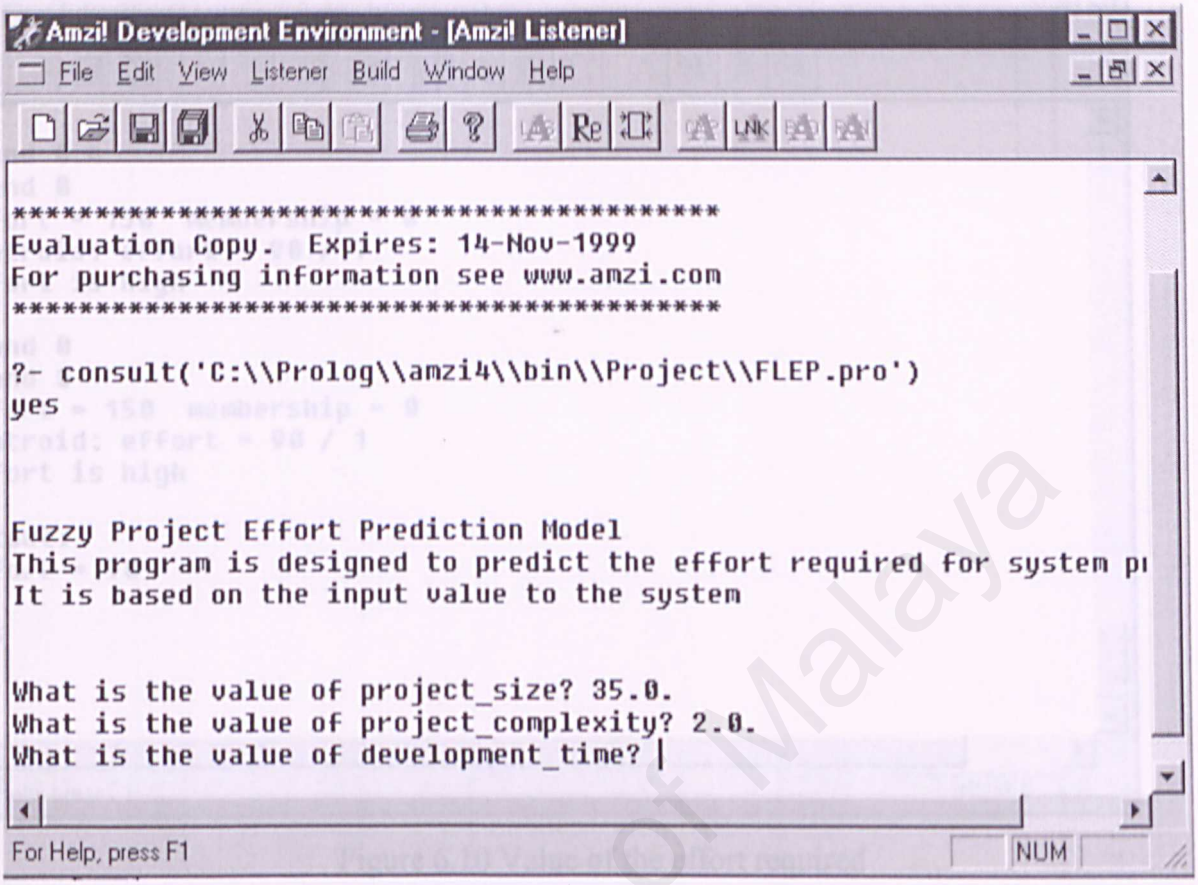

Figure 6.9 Value For Project Complexity and Development Time

The effort value is the output of the program. Defuzzification is done by computing the centroid of the area enclosed by the member function in order to get the value of the required effort. The effort required for this example is shown in the figure 6.10.

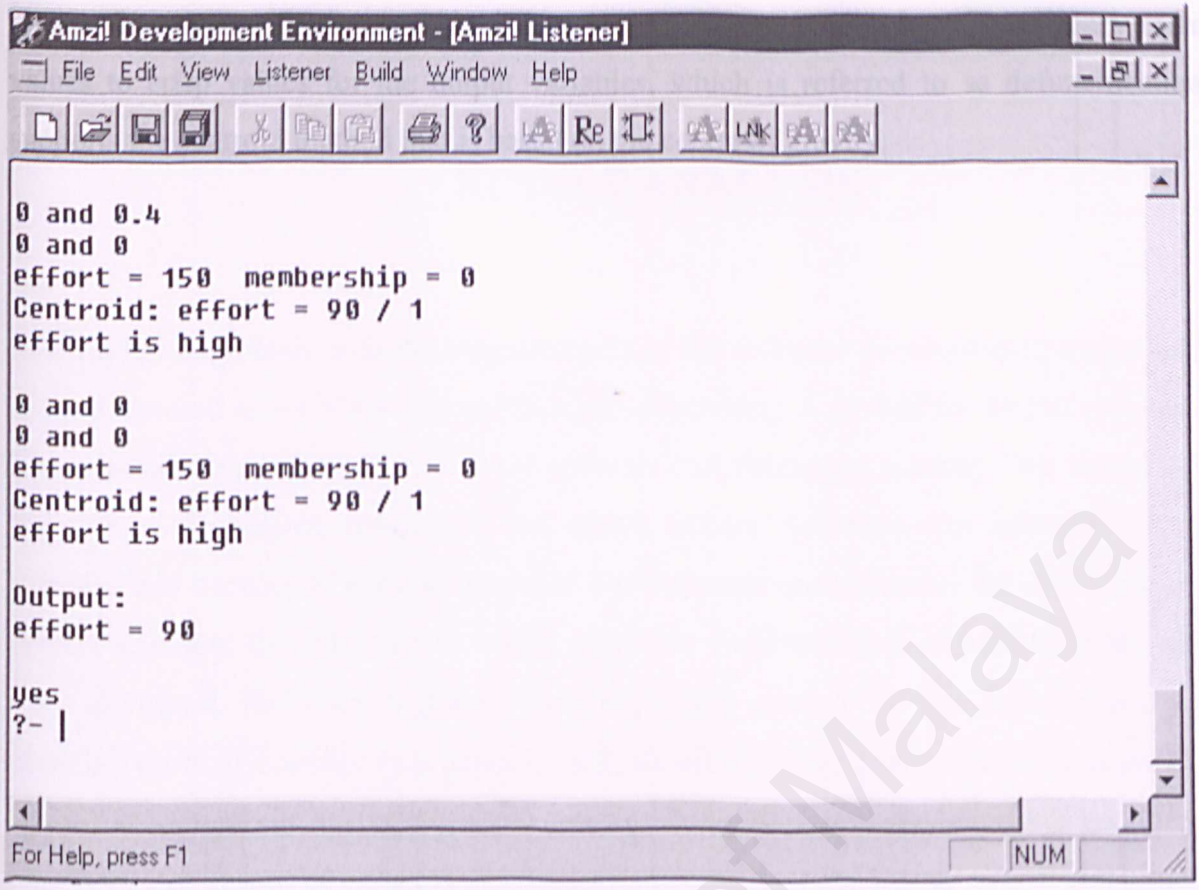

Figure 6.10 Value of the effort required

6.5 Summary

The Prolog programming language offers several opportunities to tackle the problems of software cost estimation. By exploiting certain characteristics of Prolog, the robustness of programs as well as improving the prediction process could be improved. **F**uzzy **L**ogic **E**ffort **P**rediction (**FLEP**) is a program that uses fuzzy logic concept within the Amzi! Prolog language environment. Amzi! Prolog language provides a powerful and flexible environment for programming fuzzy logic applications. It supports the concept of fuzzy sets and fuzzy rules. Fuzzy rules define relationships between different fuzzy sets as if-then rules. In FLEP, these rules are expressed using a simple syntax. In fuzzy reasoning over sets there are standard operations such as union and intersection which can be defined in terms of simple syntax operations such as

max and min. The final stage of this program is the conversion back from fuzzy membership values to crisp values for the output variables, which is referred to as defuzzification. FLEP supports the centroid method that is based on the centre of gravity.

CHAPTER 7

RESULT ANALYSIS

Software metrics deals with the measurements of the software development process and product that can be used as variables (independent and dependent) in models for project management [1]. The most common type of model is the software cost estimation models. This model is based on the size of the project, complexity and others metrics. Software cost estimation supports the planning and tracking of software projects. The estimator must estimate the effort required for the project to enable the managers to assess important quantities such as product costs, quality and time to market. Software engineers have expressed concern on the existing cost estimation models over their inability to accurately estimate effort. This concern has become more pressing when the researchers observed that no existing cost estimation models could estimate the true cost with any degree of certainty [2]. Analysis reveals that the inputs to cost estimation models are influenced by many factors such as experience of a programmer, size and complexity of the project. As a result, further analysis has been made in gaining a better understanding of the cost estimation process as well as evaluating the software cost estimation models. This chapter compares the performance of the fuzzy logic model against the three existing models in Kemerer's [8] validation study namely:

- COCOMO
- Function Points
- SLIM

Two research questions are addressed in this chapter:

1. Are the existing software effort estimation give reasonable results?
2. Can fuzzy logic be used as alternative technique in effort estimation?

In section 7.1 of this chapter, we will focus on the definition of a good estimate and we will look on several measures of estimation accuracy. Section 7.2 briefly discusses the selection of four models. The project data used for the evaluation, which leads directly into the results for each model, will be discussed in section 7.3. Section 7.4 presents the comparison of the four models and finally section 7.5 gives a summary on the issues that have been discussed in this chapter.

7.1 Methods for Measuring

Validation of a prediction system is the process of establishing the accuracy of the prediction system by comparing model performance with known data points in a given environment. There are many methods for measuring the performance of effort estimation models and all of them have limitations and strengths. We denote the actual effort expended on a software project by E and the effort predictive by a model by $\bar{E}$.

7.1.1 The Relative Error (RE) And Mean Relative Error ($\overline{RE}$)

Relative error is defined by [3]

$$RE = (E - \bar{E}) / E$$

RE measurement can be negative or positive. If $\bar{E} > E$ then $RE < 0$, while if $\bar{E} < E$ then $RE > 0$. When $RE > 0$ it must be between 0 and 1, while when $RE < 0$, it is essentially unbounded in magnitude. We can define the mean relative error of a set of n projects by the formula [3]

$$\overline{RE} = \frac{1}{n} \sum_{i=1}^{n} RE$$

A good representation model of effort estimation will lead to small values of RE and generally to a small $\overline{RE}$. However, it is possible that large positive RE_s can be balanced by large negative RE_s. At the same time, a small $\overline{RE}$ may not imply that a model is a good one. Thus, this measure is not that useful in practice.

7.1.2 The Magnitude and The Mean Magnitude of the Relative Error.

The magnitude of relative error (MRE) is a measure of the difference between the actual values of man-month estimates and the estimated values of man-month estimates. Through the view of the problem associated with RE and RE in section 7.1.1, we define the magnitude relative error MRE as [3]

$$MRE = |RE| \quad (E - \overline{E}) / E$$

The smaller the value of MRE the better the prediction is. The $\overline{MRE}$ is the mean value for magnitude relative error over all observations in the sample. For a software development project management relative errors are often a better judge of the model's accuracy. We can define the mean magnitude relative error of a set of n projects by the formula [3]

$$\overline{MRE} = \frac{1}{n} \sum_{i=1}^{n} MRE$$

If the $\overline{MRE}$ value is small, then the model produces an average of a good set of predictions. However, Conte, Dunsmore and Shen [3] suggest that an acceptable level for mean magnitude of relative error is something less than or equal to 0.25.

7.1.3 Prediction At Level r [PRED (r)]

The PRED measure provides an indication of overall fit for a set of data points, based on the MRE values for each data point. Therefore we define the measure as [3]

$$\text{PRED}(r) = \frac{k}{n}$$

where r is the selected threshold value for MRE, k is the number of data points with MRE less than or equal to r and n is the total number of data points. For example, if PRED (0.25) = 30%, then 30% of the predicted values falls within 25 % of their actual values. Conte, Dunsmore and Shen suggest that an estimation technique is acceptable if PRED(0.25) is at least 0.75 [3].

7.2 Background to the Models

Many models have been developed to estimate software development effort. Most of them are parametric in nature, which predict the development effort using a formula of fixed form that is parameterized from historical data [8]. The following are the brief description of three such models that were highlighted in a previous study by Kemerer [8] including the fuzzy logic model.

7.2.1 COCOMO

The Constructive Cost Model (COCOMO) was developed by Barry Boehm [5,8]. This model predicts the effort and duration of the project based on inputs relating to the size of the system and a number of "cost drivers". There are three types of COCOMO models; the Basic, Intermediate and Advanced models. Boehm identified three levels of development modes

namely, organic, semi-detached and embedded. In the simplest term, COCOMO is based on the following two equations [5,8]:

$$MM = a * KDSI * b$$

$$TDEV = c * MM * d$$

where

MM is the effort in man-months

KDSI is the number of thousand-delivered source instructions

TDEV is the development time

Coefficients a, b, c and d are dependent upon the 'mode' of development in the following modes:

- **Organic** - projects involve small teams working in familiar and stable environment.
- **Semi-detached** – mixture of experience within project teams. In between on embedded modes.
- **Embedded** – projects that are developed under tight constraints, innovative and have a high volatility of requirements.

The values of a, b, c and d are shown in the table 7.1.

Table 7.1

Values Of the Coefficient

Development Mode	a	b	c	d
Organic	3.2	1.05	2.5	0.38
Semi-detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

In order to improve the COCOMO Basic model, Boehm refined the equation above to adjust the nominal effort obtained from the model by the influence of 15 cost drivers. For example, if the reliability of the software is very high, a factor rating of 1.40 can be assigned to that driver. Once all the factors for each driver have been determined, they are multiplied to get an effort adjustment factor (EAF). This version is the intermediate model. The COCOMO detailed model is similar with the intermediate model except that the project is divided into four phases namely, Product Design, Detailed Design, Coding/Unit Test and Integration / Test. The 15 cost drivers are estimated and applied to each phase separately rather than as a whole [5,8].

7.2.2 Function Points

Function Points [5,8,9] measure was invented by A.J. Albrect in the 1970's. This model attempts to measure the software from the standpoint of the consumer of the software product in the early development life cycle process. Function Points is possible to estimate the level of effort required on a software-development project. There are two steps in counting the function points known as:

(1) Counting the user functions
(2) Adjusting for processing complexity

Computation of the total function points is based on the five user functions categories. These user functions categories include external input types, external output types, logical internal file types, external interface file types and external inquiry types.

1. **External input types** – Inputs are defined as screens of forms through which users or other programs add new data or update existing data.
2. **External output types** – Outputs are screens or reports namely, produced for human use or for use by other programs.

3. **Logical internal file types** – File types are logical collections of file, which the application modifies or updates.
4. **External interface file types** – Interfaces are files shared with other programs.
5. **External inquiry types** – Inquiries are screens that allow the user to interrogate an application.

Albrecht recognized that the effort required to providing a given level of functionality can vary depending on the environment [9]. He also proposed a list of 14 technical complexity factors, which can be rated on a scale from 0 to 5. The Technical Complexity Factors are:

1. Data communication
2. Distributed functions
3. Performance
4. Heavily used system
5. Transaction rate
6. On-line-data entry
7. End-user efficiency
8. On-line update
9. Complex processing
10. Reusability
11. Installation ease
12. Operational ease
13. Multiple sites
14. Facilitate change

The next step is to sum all the processing complexity points assigned [9]. This number is then multiplied by 0.01 and added to 0.65 to obtain weighting as follows [9]:

$$PCA = 0.65 + 0.01 \sum_{i=1}^{14} c_i$$

where PCA is the processing complexity adjustment ($0.65 \leq PCA \leq 1.35$) and c_i is the complexity factors ($0 \leq c_i \leq 5$). This factor is then used in the equation

$$\mathbf{FP = FC(PCA)}$$

where

FP is the function points

FC is the previously computed function counts.

7.2.3 SLIM

Putnam [3,8,18] developed an early model known as SLIM which estimates the cost of software using lines of code as the major inputs. Putnam assumed that resource consumption, including personnel, varies with time and can be modeled by the Rayleigh equation of this form [3,8,18] to produce its effort estimates. The Rayleigh equation is shown below:

$$\mathbf{y' = 2Kat\ exp\ (-at^2)}$$

where

y' is the man-loading (man-years of effort per year)

K is the total man-years of effort for the entire project or the life cycle effort

t is the project duration time and

a is the constant of proportionality.

The Putnam SLIM model is based on earlier empirical observations by Norden [3,8]. According to Putnam's observation, software projects often behave in accordance with Rayleigh-Norden forms [3,8]. He related the system attributes, number of files, modules and reports to the manpower. He understands exactly what the software development process consists of over its

life cycle, maintaining a data base that reflects the history of actual software development costs and developing the most cost-effective allocation of resources to different phases of software.

7.2.4 Fuzzy Logic Model

A significant motivation of using fuzzy logic is the ability to estimate required effort in the early process of software development. Fuzzy logic was developed with the logic concept, where in this approach the entities in the real world simply do not fit into neat categories. For example, a project is not small, medium or large. It could be in between, especially a large project [1,31,32]. This case is represented by a degree of membership of a particular linguistic category. As shown in the figure 7.1, a system with 162 entities belongs to the class of medium projects to a degree of 0.4 and the class of large projects to a degree of 0.6 [1].

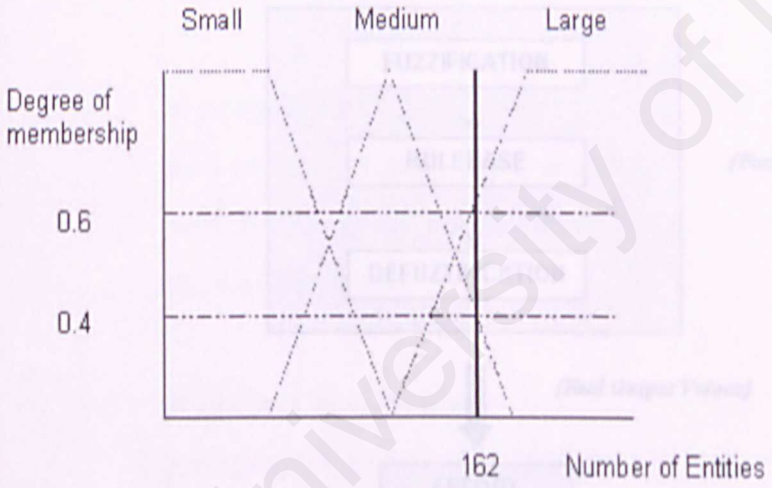

Figure 7.1 Fuzzy Set Membership

Fuzzy set theory and fuzzy logic were proposed by Zadeh in 1965. The basic principle of this theory is that, a set of membership values are assigned to observations in the range 0.0 to 1.0 where a value of 0.0 represents absolute false and 1.0 represents absolute truth. The value in-between represents a degree of partial truth.

For example, take the statement [32]:

"Project X is very small"

If the module contains say 50 lines of code, we might decide to assign a truth-value of (0.7) to this statement or in other words we believe the project to be "more or less, very small".

In the example shown in figure 7.2, three input variables, namely, the size of the project, complexity of the project and total development time, are used to measure the effort.

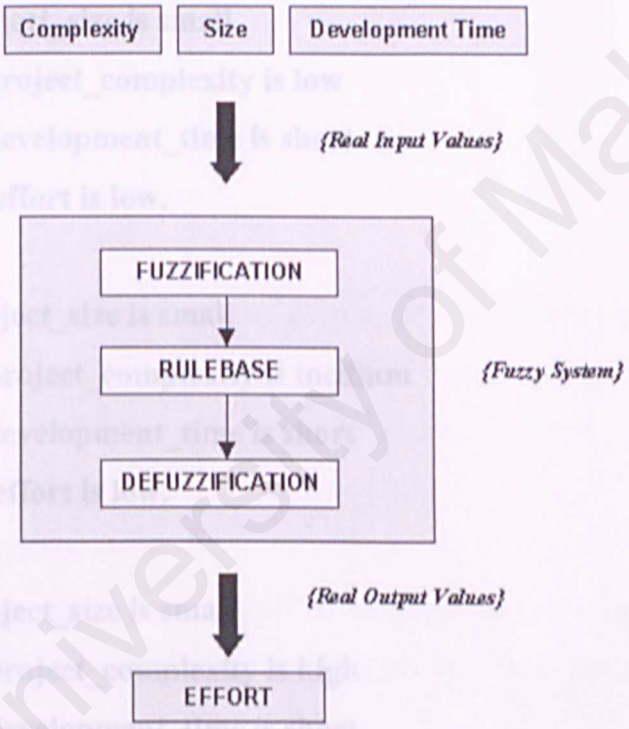

Figure 7.2 Fuzzy System

Measurement of effort is the output of this flow. The fuzzy system consists of three components:

1. Input membership level.
   The fuzzification process will convert the input variables into appropriate fuzzy sets and determine their membership in those sets.

2. Fuzzy rulebase level.

   A rule induction system will replace the crisp input variables value with fuzzy inputs. An inference engine will then use the fuzzy rulebase to map inputs to a fuzzy output. In other words the inputs will be defuzzified into one crisp value. The fuzzy rules produced in the system are listed below and are represented by an antecedent and a consequence, i.e.

   IF (antecedent) THEN (consequence)

   if project_size is small
   and project_complexity is low
   and development_time is short
   then effort is low.

   if project_size is small
   and project_complexity is medium
   and development_time is short
   then effort is low.

   if project_size is small
   and project_complexity is high
   and development_time is short
   then effort is medium.

   if project_size is small
   and project_complexity is low
   and development_time is medium
   then effort is low.

if project_size is small
and project_complexity is medium
and development_time is medium
then effort is medium.

if project_size is small
and project_complexity is high
and development_time is medium
then effort is medium.

if project_size is small
and project_complexity is low
and development_time is long
then effort is low.

By referring to the fuzzy rules above, the most likely outcome for a small size project with low complexity and longer development time is lower effort will be required. This rule is common and practical in nature.

3. Defuzzification level.

   The fuzzy output are defuzzified to generate the crisp output, which is the effort. Many methods exist for defuzzification, but most popular methods are centroid and center of maxima.

7.3 Project Data and Results

The source for the project data for this project is the actual project data from a research paper of "An Empirical Validation Of Software Cost Estimation Models" by Chris F. Kemerer [8]. He conducted empirical analysis between COCOMO, SLIM and Function Points models on 15

projects' data. These projects deal with business applications which dominant proportions of them are written in the COBOL language.

Table 7.2 [8] shows the background data on the projects. The "Months" are project duration in calendar months. The man-months (MM) refer to the total number of actual hours on the project through implementation. The effort measure was the only output of the models that was evaluated.

Table 7.2
Project Background

Project	Software	Hardware	Months	MM	KSLOC
1	Cobol	IBM 308X	17	287.00	253.60
2	Cobol	IBM 43XX	7	82.50	40.50
3	Cobol	DEC VAX	15	1,107.31	450.00
4	Cobol	IBM 308X	18	86.90	241.40
5	Cobol	IBM 43XX	13	336.30	449.90
6	Cobol	DEC 20	5	84.00	50.00
7	Bliss	DEC 20	5	23.20	43.00
8	Cobol	IBM 43XX	11	130.30	200.00
9	Cobol	IBM 308X	14	116.00	289.00
10	Cobol, Natural	IBM 308X	5	72.00	39.00
11	Cobol	IBM 308X	13	258.70	254.20
12	Cobol	IBM 43XX	31	230.70	128.60
13	Cobol	HP 3000	20	157.00	161.40
14	Cobol	IBM 308X	26	246.90	164.80
15	Natural	IBM 308X	14	69.90	60.20
Mean			14.3	219.25	186.57

The data used here is mainly to illustrate the comparison of estimated effort to actual effort. There are several ways in evaluating the estimated effort. A simple approach is to look at the difference between estimated effort and actual effort. Errors can be either underestimates, where $MM_{est} < MM_{act}$ or overestimate where $MM_{est} > MM_{act}$. Both of these errors can lead to large underestimates that cause the project to be understaffed, and only when the deadline approaches, project management will be tempted to add new staff members [8]. On the other hand, overestimates can lead the project to be less productive.

From the view of the both types of errors, the magnitude relative error (MRE) and mean magnitude relative error ($\overline{MRE}$) can be used in measuring the performance of the four models. The analysis of this chapter concentrates on the models' average performance over the entire set of project's data.

7.3.1 COCOMO Results

Table 7.3 [8] shows the results for the three COCOMO models. In this table, "BAS" refers to the Basic Model, "INT" refers to the Intermediate Model and "DET" refers to the Detailed Model.

Table 7.3

COCOMO Data

Project Number	Actual MM	COCOMO BAS	COCOMO BAS (MRE)	COCOMO INT	COCOMO INT (MRE)	COCOMO DET	COCOMO DET (MRE)
1	287.00	1,095.10	2.81	917.56	2.19	932.96	2.25
2	82.50	189.40	1.29	151.66	0.83	151.19	0.83
3	1,107.31	5,497.40	3.96	6,182.65	4.58	5,818.75	4.25
4	86.90	1,222.30	13.06	558.98	5.43	566.50	5.51
5	336.30	1,466.00	3.35	1,344.20	2.99	1,316.04	2.91
6	84.00	393.60	3.68	313.36	2.73	312.24	2.71
7	23.20	328.40	13.15	234.78	9.11	234.51	9.10
8	130.30	925.30	6.10	1,165.70	7.94	1,206.17	8.25
9	116.00	3,231.00	26.85	4,248.73	35.62	4,577.62	28.46
10	72.00	181.60	1.52	180.29	1.50	181.36	1.51
11	258.70	1,482.20	4.72	1,520.04	4.87	1,575.68	5.09
12	230.70	691.00	1.99	558.12	1.41	584.37	1.53
13	157.00	891.20	4.67	1,073.47	5.83	1,124.36	6.16
14	246.90	511.00	1.06	629.22	1.54	663.84	1.68
15	69.90	295.30	3.22	133.94	0.91	130.72	0.87
Mean (absolute values)	219.25	1,226.72	6.09	1,280.85	5.83	1,291.75	6.07

The MRE columns are calculated by dividing the difference between the actual and the estimate man-months by the actual man-months. The $\overline{MRE}$ for Basic COCOMO model is 6.09, Intermediate COCOMO model is 5.83 and the Detailed COCOMO model is 6.07.

7.3.2 SLIM Results

Table 7.4 [8] shows the results for the Function Points model. Also shown are the actual man-months and the SLIM man-month estimate. The $\overline{MRE}$ for SLIM model is 0.58.

Table 7.4

SLIM Data

Project Number	Actual MM	SLIM MM estimate	SLIM (MRE)
1	287.0	3,857.8	12.44
2	82.5	100.1	0.214
3	1,107.31	11,982.0	9.82
4	86.9	2,017.2	22.21
5	336.3	3,382.0	9.06
6	84.0	262.5	2.13
7	23.2	106.3	3.58
8	130.3	1,224.6	8.39
9	116.0	1,454.1	11.53
10	72.0	235.7	2.27
11	258.7	1,623.0	5.27
12	230.7	513.3	1.23
13	157.0	3,119.8	18.87
14	246.9	380.3	0.54
15	69.9	643.8	8.21
Mean (absolute values)	219.25	2,060.17	7.717

7.3.3 Function Points Results

Table 7.5 [8] shows the results for the Function Points model. The $\overline{MRE}$ for the Function Points model is 1.02.

Table 7.5

Function Points Data

Project Number	Actual MM	Function Points MM Estimate	Function Points error (MRE)
1	287.0	344.30	0.19
2	82.5	92.13	0.11
3	1,107.31	731.43	0.33
4	86.9	192.03	1.20
5	336.3	387.11	0.15
6	84.0	61.58	0.26
7	23.2	-52.60	3.26
8	130.3	264.68	1.03
9	116.0	477.81	3.11
10	72.0	-2.83	1.03
11	258.7	484.24	0.87
12	230.7	192.21	0.16
13	157.0	157.36	0.02
14	246.9	390.63	0.58
15	69.9	282.91	3.04
Mean (absolute values)	219.25	266.87	1.02

7.3.4 Fuzzy Logic Results

Table 7.6 shows the results for the Fuzzy Logic model and the $\overline{MRE}$ result is 0.59.

Table 7.6

Fuzzy Logic Data

Project Number	Actual MM	Fuzzy Logic MM Estimate	Fuzzy Logic (MRE)
1	287.00	139.47	0.51
2	82.50	130.00	0.17
3	1,107.31	150.00	0.86
4	86.90	135.71	0.56
5	336.30	150.00	0.55
6	84.00	90.00	0.07
7	23.20	90.00	2.88
8	130.30	150.00	0.15
9	116.00	150.00	0.29
10	72.00	90.00	0.25
11	258.70	150.00	0.42
12	230.70	130.06	0.44
13	157.00	130.93	0.17
14	246.90	118.69	0.52
15	69.90	150.00	1.15
Mean	219.25	130.32	0.59

Fuzzy logic model performs well via the $\overline{MRE}$ test. One possible reason is that fuzzy logic approach can overcome the difficulties implicit in obtaining subjective input parameters. The input variable used for the effort prediction at the beginning of the project is very subjective and vague.

7.4 Comparison of Results

Accuracy is the criteria used to evaluate estimation models. The accuracy of an estimation model is important because the success of a software development project is directly affected by the effort estimation. Inaccurate estimates can lead to bad impact on the project scheduling and level of staffing. In order to analyze, the performance of the four models, we need to compare the results of the Mean Magnitude of Relative Error ($\overline{MRE}$). The $\overline{MRE}$ is a measure of models accuracy, the lower the score the greater the accuracy. Each of this model's results is shown in the figure 7.3.

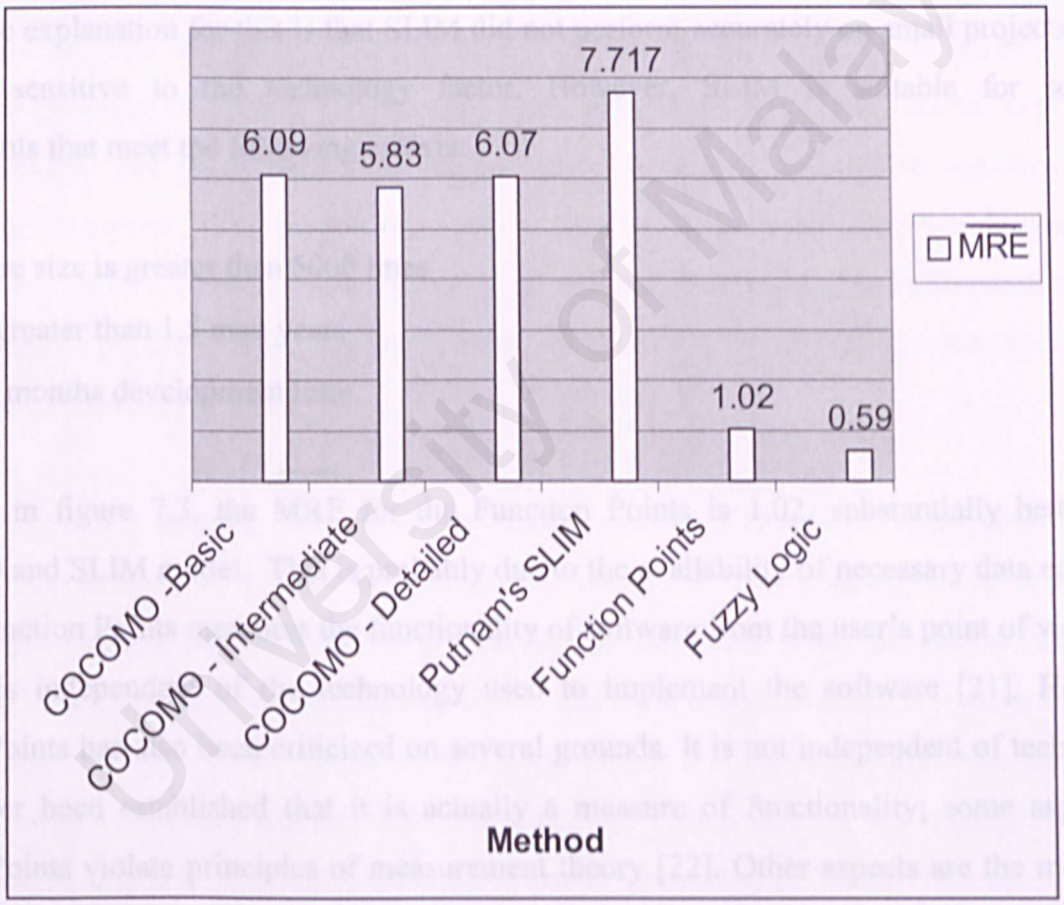

Figure 7.3 Comparison of Models

From the above table, all of the COCOMO models did poorly according to the $\overline{MRE}$ results. The $\overline{MRE}$ result for the Basic COCOMO model is 6.09, Intermediate COCOMO model is 5.83 and the Detailed COCOMO model is 6.07. This implies that using a large number of cost drivers in estimating effort may not be independent. Kitchenham [3,19] demonstrates that there is a relationship between two of the COCOMO cost drivers based on projects in the COCOMO data set and points out the relationships between input parameters will make a model unstable.

The performance of the SLIM model is also poor according to the $\overline{MRE}$ result, which is 7.717. The errors are all biased and effort is overestimated in all 15 cases [19]. According to Kemerer the possible explanation for this is that SLIM did not perform accurately on small projects and is extremely sensitive to the technology factor. However, SLIM is suitable for software developments that meet the following criteria:

- Software size is greater than 5000 lines
- Effort greater than 1.5 man years
- Over 6 months development time

As shown in figure 7.3, the $\overline{MRE}$ for the Function Points is 1.02, substantially better than COCOMO and SLIM model. This is probably due to the availability of necessary data early in a project. Function Points measures the functionality of software from the user's point of view, in a way that is independent of the technology used to implement the software [21]. However, Function Points has also been criticized on several grounds. It is not independent of technology; it has never been established that it is actually a measure of functionality; some aspects of Function Points violate principles of measurement theory [22]. Other aspects are the method is subjective and hard to automate.

Compared to the other models, the fuzzy logic model seems to show good performance in terms of $\overline{MRE}$ measures. Cost estimation is seldom accurate especially in the early development life cycle when precise values for metrics are not available. Our analysis indicates that the

performance of the four models depends on the input variables used for the cost estimation at the beginning of the project. The fuzzy logic technique can be applied in the early development life cycle when input variables are defined based on perception and intuitive. A project manager may say that the 'size of the project is very high" based on his perception of size. The technique requires only modest inputs and comes up with an expected value of effort. Given the other advantages of a fuzzy logic model, this suggests that there is a place for it in the field of software metrics.

7.5 Summary

The software projects should be grounded on a good prediction of the effort development to improve the planning of project management. This implies on numerous factors, used in the effort prediction. The measurement of the influence factors is difficult to measure at the beginning of the project. Thus, project managers have to use their expert judgement in measuring these influence factors. After the four software cost models are examined and compared with the use of Kemerer's [8] set of data, it seems appropriate to answer the addressed research questions posed in the introduction of this chapter.

The validation of the existing cost estimation models, namely, COCOMO, SLIM, and Function Points will help answer the first question. The equations of the three models must be sensitive to the project attributes of the products, the process and the resources. However, the cost factors of COCOMO, Function Points and SLIM have shown that there is no correlation between these cost factors' values and the effort. Some of the cost factors are confusing and are difficult to determine the values in the early development life cycle. This points out that a project manager needs to specify only important input variables required for the cost estimation process.

The second research questions concerned on the use of fuzzy logic as an alternative technique in effort estimation. Fuzzy logic allows representation of input variables as vague (or probabilistic depending on your preference) membership functions and provides operations on fuzzy sets. Membership functions represent the degree of matching between values and the fuzzy set. Given

the input values for the independent variables their membership functions are mapped into certain degree of memberships. They in turn activate rules using AND operations and defuzzification is used to obtain final value from the combined membership functions. An exact prediction is not expected (and possibly not even realistic) but by using the classifications with fuzzy logic linguistic variables better prediction can be obtained. Thus, fuzzy logic approach provides a promising approach to improving the prediction.

The idea of using fuzzy logic for cost estimation has been outlined in this chapter. The motivation for this has been the difficulties faced by software metricians in terms of avoiding premature and costly commitment, using all available knowledge and having only small data sets to work with [1]. Compared to other models, the fuzzy logic model shows good performance.

CHAPTER 8

CONCLUSION AND RECOMMENDATIONS

This chapter is divided into two sections. The first section summarizes the ideas of the software cost estimation that have been discussed in this thesis. In the second section, recommendations, namely, applicable for further research are discussed.

8.1 Review and Contributions of Work

In this thesis, fuzzy logic approach to software cost estimation is investigated. Software cost estimation is a common application in the software metrics. In general, software metrics refers to the whole field of measurement in the software development process, as it measures the processes (which normally entail a time duration), products (the process outputs) and resources (the process input and which may include personnel, materials and tools). It is used for planning (make predictions), monitor (check for deviations between actual and expected) and control the software development process. Planning is vital to ensure the successful management of systems development activities where it requires prediction process. The purpose for which a prediction is required is the central influence on what measures are to be predicted. For example, predictions are required in the software cost estimation to observe the effort required for the development of a software project before work begins. The effort is a measure of labor cost in person-months units.

Several cost estimation models such as COCOMO, Function Points and SLIM have been proposed. There are some questions in validation of these models applied to a wide range of projects. It is suggested that a model is acceptable if 75 percent of the predicted values fall within 25 percent of their actual values [3]. Unfortunately, these models are insufficient based on this

criteria. Several reasons why these models have fallen short of their goals and these include model structure and input values.

Even though, most researchers agree that size is the primary determinant of effort, the relationship between size and effort is unclear [19]. Most empirical studies express effort as a function of size with an exponent **b** and **b** is included as an adjustment factor. However, the value of **b** is varying from different data sets as shown in the table 8.1 [19].

Table 8.1

Comparison of Effort Equation Adjustment Factor

Model	Adjustment Factor
COCOMO (organic)	1.05
COCOMO (semi-detached)	1.12
COCOMO (embedded)	1.20
Putnam	1.286

Software cost estimation inputs values are often not known with reasonable certainty at the time of estimation. Inputs to these models such as software reliability and experience of the programmers are difficult to measure. Furthermore, some of the models such as COCOMO and Functions Points assume that the inputs are independent but this is not the case in practice. Many of the inputs affect each other and also extremely subjective. As such, it is difficult to ensure that different people assess the factors consistently.

In order to overcome the limitations of the existing models, fuzzy logic approach in cost estimation process is adopted. In this thesis, we have proceeded along this path by providing the fuzzy logic basis and concept, which is necessary for the development of fuzzy logic effort prediction program. Major contributions of this thesis are as follows:

1. Fuzzy logic linguistic variable concept is applied in capturing the input values (such as size, complexity and so on) to cost estimation process. Based on this idea, the linguistic variable is capable of capturing the subjective and uncertained input values such as the size, complexity and duration time of a particular project for better prediction. Consequently, the values for these linguistic variables can either be small, medium or large (size), low, medium or high (complexity) and short, medium or long (duration time). Through this linguistic approach the subjective and uncertained inputs values can easily be measured.

2. Rule-based logic is used to capture human expertise in estimating the effort. For example, consider the rule:

IF project size is high THEN effort is high

Where, project size is the linguistic variable, effort is an output linguistic variable and high is the linguistic values (membership functions defined on project size fuzzy set). Fuzzy rule is used as a method of capturing human assessment of how to relate the effort to the input (size). Normally, we use our intuitive knowledge, as larger projects would seem to require more effort than smaller projects.

3. Incorporate the fuzzy system model in developing the fuzzy logic approach to effort estimation process. Fuzzy system model consists of (1) fuzzification of the terms that appear in the conditions of rules, (2) inference from fuzzy rules and (3) defuzzification of the fuzzy terms that appear in the conclusions of rules. In the fuzzification process, the membership functions defined on the input variables are applied to their actual values to determine the truth for each rule premise. While in the inference process, the truth-value for the premise of each rule is computed and applied to the conclusion part of each rule. This will results in one fuzzy set to be assigned to each output variable for each rule and through the composition process, all the fuzzy sets are combined to form a single fuzzy set for output variable (in this case is the effort). The last process is the defuzzification where the value of the output fuzzy set will be converted into a single crisp value.

4. Related to the point 3 above, we develop Fuzzy Logic Effort Prediction (FLEP) program using Amzi! Prolog. FLEP uses three inputs, namely, size, complexity and duration time.

5. A comparison study between the COCOMO, Function Points, Slim and fuzzy logic is conducted. The purpose of this study is to evaluate whether the models give reasonable results and also to justify if the fuzzy logic can be used as alternative technique in effort estimation. We analyze these models based on the Mean Magnitude Relative Error ($\overline{MRE}$) on Kemerer's 15 project's data sets [8]. From the analysis, fuzzy logic model seems to show good performance compared to other models. Since many of the input are either difficult to quantify (complexity) or only known to a rough degree (size), the use of fuzzy logic seems appropriate. The project manager would be able to make consistent classifications of project by using fuzzy linguistic variables. Meanwhile, the performance of the three models: COCOMO, Function Points and SLIM are poor according to the $\overline{MRE}$ measurement.

8.2 Further Research Work

Although fuzzy logic offers considerable promise to software metrics, there is considerable scope of further research work arising from this thesis. These include the following:

1. The use of neuro-fuzzy system as another technique to software cost estimation should be investigated. A neuro-fuzzy system is a fuzzy system that uses a learning algorithm derived from neural network theory to determine its parameters (fuzzy sets and fuzzy rules) by processing data [31,32,44,45,46,47,48]. Generally, a neuro-fuzzy system can be viewed as a 3-layer feed forward neural network. The first layer represents input variables, the middle (hidden) layer represents fuzzy rules and the third layer represents output variables. Fuzzy sets are encoded as connection weights. Hence, this would be a good topic for future research.

2. Case based reasoning is a method of storing observations on previous projects such as effort required in implementing the project and programming platform [31,32,45,46]. When faced with a new observation it is possible to retrieve the closest stored observations. Application of case based reasoning to cost estimation can be an interesting future direction of research.

LIST OF REFERENCES

[1] A. Gray and S. MacDonell. "Applications of Fuzzy Logic to Software Metrics Models for Development Effort Estimation", In Proceedings of the 1997 Annual Meeting of the North American Fuzzy Information Processing Society (NAFIPS), New York, pp. 394-399, 1997.

[2] T. Mukhopadhyay and S. Kekre. "Software Effort Models For Early Estimation Of Process Control Applications". *IEEE Transactions on Software Engineering*, vol., 18 pp.915-924, 1992.

[3] N. Fenton. *Software Metrics, a Rigorous Approach*. London:Chapman & Hall, 1991.

[4] S.L. Pfleeger. *Software Engineering : Theory and Practice. New Jersey*, Prentice Hall, 1998.

[5] B.W. Boehm, "Software Engineering Economics," *IEEE Trans. Software Eng.*, vol. SE-10, pp. 4-21, Jan. 1984.

[6] S.D. Conte, H.E. Dunsmore, and V.Y Shen, *Software Engineering Metrics and Models*. Menlo Park, CA: Benjamin/Cummings, 1986.

[7] F. Walkerden and R.Jeffery. "Software Cost Estimation : A Review of Models, Process and Practice", *Advances in Computers*, vol.14, pp.61-123, 1997.

[8] C.F Kemerer, "An Empirical Validation Of Software Cost Estimation Models", *Commun. ACM*, vol. 30, pp. 416-429, May 1987.

[9] H. Zuse. *A Framework of Software Measurement*, New York, Walter de Gruyter, 1998.

[10] R. Jager. *Fuzzy Logic in Control*. PhD's Thesis, Delft University of Technology, Netherlands, 1995.

[11] G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic*. New Jersey, Prentice Hall, 1995.

[12] G.J Klir and B.Yuan. *Fuzzy Sets, Fuzzy Logic and Fuzzy Systems*. Singapore, World Scientific Publishing, 1996.

[13] A. Lotfi. *Learning Fuzzy Inference Systems*. PhD's Thesis, University of Queensland, Australia, 1996.

[14] R. R. Levary and C. Y. Lin. "Modelling The Software Development Process Using An Expert Simulation System Having Fuzzy Logic", *Software-Practice and Experience*, vol., 21, pp. 133-148, 1991.

[15] R.I. Kilgour, A.R. Gray, P.J. Sallis and S.G. MacDonell. "A Fuzzy Logic Approach To Computer Software Source Code Authorship Analysis". Submitted to The Fourth International Conference on Neural Information Processing – The Annual Conference of the Asian Pacific Neural Network Assembly (ICONIP'97).

[16] S. N. Mohanty. "Software Cost Estimation: Present and Future", *Software-Practice and Experience*, vol., 11, pp.103-121, 1981.

[17] K.Pillai and V.S.S. Nair, " A Model for Software Development Effort and Cost Estimation", *IEEE Transactions on Software Engineering*, vol.,23, pp. 485-597, 1997.

[18] K. Srinivasan and D. Fisher. "Machine Learning Approaches To Estimating Software Development Effort", *IEEE Transactions on Software Engineering,* vol.,21 pp. 126-137, 1995.

[19] S.D. Conte *et al*., "Software Effort Estimation and Productivity", *Advances In Computers*, vol., 24, pp.1-59, 1985.

[20] A. Abran and P.N. Robillard. "Function Points Analysis: An Empirical Study of Its Measurement Processes", IEEE Transactions on Software Engineering, vol. 22, pp. 895-909, 1996.

[21] A.J Albrecht and J. Gaffney, "Software Function, Source Lines Of Code And Development Effort Prediction: A Software Science Validation," IEEE Trans. Software Eng., vol. SE-9, pp. 69-75, July 1986.

[22] C.R. Symons, "Function Point Analysis: Difficulties And Improvements," IEEE Trans. Software Eng., vol. SE-14, pp. 2-11, Jan. 1988.

[23] T. Onisawa and J. Kacprizyk. *Reliability and Safety Analyses under Fuzziness*. Germany, Physics-Verlag Heidelberg, 1995.

[24] P.P Wang and S.K. Chong. *Fuzzy Sets : Theory And Applications To Policy Analysis and Information Systems*. New York, Plenum Press, 1980.

[25] C. Aschmann. *Fuzzy Systems for Information Processing*, Tokyo, Ohmsha, 1995.

[26] A. Kandell and M. Schneider. "Fuzzy Sets and Their Applications to Artificial Intelligence", *Advances In Computer*, vol., 28, pp. 69-103, 1989.

[27] B. Turksen. "Semantic Uncertainty of the Fuzzified Laws of Logic", *IEEE Expert*, pp. 34-35, August, 1994.

[28] R. Forsyth. *Expert Systems : Principles and Case Studies*. New York, Chapman and Hall, 1986.

[29] Software Cost Estimation Web-Site (SCEW), April 10, 1999. http://www.ecfc.u-net.com/cost/index.htm.

[30] Gray, Overview of the Development Effort Estimation Project, June 28, 1999. http://divcom.otago.ac.nz/COM/INFOSCI/SMRL/projects/RESEARCH/effort.htm.

[31] S.G. MacDonell and A.R Gray. "Alternatives To Regression Models For Estimating Software Projects", *In Proceedings of the IFPUG Fall Conference*, Dallas TX, IFPUG 279.1-279.15, 1996.

[32] A.R. Gray and S.G. MacDonell. "A comparison of model building techniques to develop predictive equations for software metrics", *Information and Software Technology*, to appear, 1997.

[33] S.G. MacDonell and A.R. Gray. "A Comparison Of Modeling Techniques For Software Development Effort Prediction", In Proceedings of the 1997 International Conference on Neural Information Processing and Intelligent Information Systems, Dunedin, New Zealand, pp. 869-872, 1997.

[34] C.V Negoita. *Expert Systems and Fuzzy Systems*, California, Benjamin/Cummings, 1985.

[35] J. Durkin, *Expert Systems : Design and Development*. New Jersey: Prentice Hall, 1994.

[36] L.H. Putnam and W. Myers. *Measures for Excellence : Reliable Software on Time, Within Budget*. New Jersey, Prentice Hall, 1992.

[37] W.S.Humprey. *A Discipline For Software Engineering*. Reading, Addison-Wesley, 1995.

[38] Fuzzy Control, Fuzzy Inference, Fuzzy Rules, Linguistic Expressions, Conditions, Conclusions, June 24, 1999. http://www.ta-eng.com/industry/mforum/fuzzy/1_2.htm.

[39] E.M Bennatan. *On Time, Within Budget Software Project Management Practices and Techniques*, Canada, McGraw Hill, 1995.

[40] White Paper on Rules, Prolog and Logic Server Technology, July 4, 1999. http://www.amzi.com/prolog.htm

[41] Prolog: An Overview, July 4, 1999. http://max.cs.kzoo.edu/~acarra/prolog.html

[42] A Concise Introduction to Prolog, July 4, 1999. http://www.netaxs.com/people/nerp/prolog/prolog.html.

[43] Guide to Prolog Programming, July 4, 1999. http://kti.mft.cuni.cz/~bartak/prolog/contents.html

[44] An Evaluation of Software Cost Models, June 28, 1999. http://saturn.cs.depaul.edu/~se/SES/htmlsf98/cesar.htm

[45] C. Stricker. Evaluating Effort Prediction Systems, June 28, 1999. http://www.cba.hawaii.edu/cs/f-prom.htm.

[46] C. Zhang *et. al.*, Product and Process Cost Estimation with Fuzzy Multi-Attribute Utility Theory, June 21, 1999. http://chaos.eng.fsu.edu/~yzhang/publications/working/ting1.htm.

[47] A. C. Hodgkinson and P. W. Garratt. A Neurofuzzy Cost Estimator, June 21, 1999. http://dsse.ecs.soton.ac.uk/~ach95r/bib/nf-icse99.html.

[48] M. Brown and C.J. Harris. "The Development and Application of Neurofuzzy Systems", In Artificial Intelligence in Consumer and Domestic Products, London, UK, pp. 1-6, 1996.